Patrícia Barbosa Lecas Espada

Licenciada em Engenharia Informática

Melhoria da qualidade para modelos orientados a objectivos: o caso da abordagem KAOS

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador: Prof. Doutor Miguel Carlos Pacheco Afonso Goulão, Professor Auxiliar, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Co-orientador: Prof. Doutor João Baptista da Silva Araújo Júnior, Professor Auxiliar, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Júri:

Presidente: Prof. Doutor Nuno Manuel Ribeiro Preguiça

Vogais: Prof. Doutor José Alberto Rodrigues Pereira Sardinha

Prof. Doutor Miguel Carlos Pacheco Afonso Goulão

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2012

Melhoria da qualidade para modelos orientados a objectivos: o caso da abordagem KAOS

Para a minha mãe.

Agradecimentos

Quero aproveitar este espaço para agradecer a todas as pessoas que, directa ou indirectamente, me ajudaram na realização desta dissertação de mestrado.

Aos meus orientadores, Professores Miguel Goulão e João Araújo, por todo o apoio prestado, pelas suas sugestões e críticas, pela confiança depositada, e por todo trabalho, interesse, disponibilidade, e paciência que demonstraram ao longo da realização deste trabalho. Foi bastante prestigiante, gratificante e enriquecedor trabalhar sobre a vossa orientação. Muito obrigada pelo vosso apreço e por partilharem a vossa experiência e o vosso conhecimento.

Ao Robert Darimont, por partilhar comigo a documentação do caso de estudo *Car Park Management*, do seu trabalho no e-Learning sobre Engenharia de Requisitos Orientada a Objectivos e sobre a ferramenta Objectiver.

A todos os meus colegas e amigos, que me apoiaram durante todo o meu percurso académico. Em especial ao João Miguel Silva, João Eduardo Luís e Tiago Melo, por me terem acompanhado desde o primeiro ano de faculdade até ao momento da entrega desta dissertação, oferecendo sempre a sua força, ajuda, e companheirismo.

Ao meu namorado Pedro, que me acompanhou durante este último ano de dissertação dando me força, mostrando interesse, e ajudando-me sempre que estava ao seu alcance.

Aos meus avós Maria José e José, que foram como pais para mim e em que o seu doce carinho fez de mim uma pessoa melhor.

Ao meu pai Simplicio e ao meu irmão Bernardo, que sempre acreditaram no meu sucesso.

À minha tia Luísa e prima Joana, que são como uma segunda Mãe e uma Irmã adorada.

E em especial, à minha mãe Teresa, responsável por tudo aquilo que sou hoje e por tudo aquilo que alcancei. Estou eternamente grata pelos esforços que fez para me proporcionar a melhor educação possível. Queria agradecer desta forma, dedicando esta dissertação, por todo o seu apoio, carinho e amor de que tanto prezo e retribuo.

Resumo

As abordagens de Engenharia de Requisitos Orientada a Objectivos (EROO) foram desenvolvidas para facilitar o trabalho dos engenheiros de requisitos ao, por exemplo, oferecer mecanismos de abstracção (isto é, os objectivos do sistema) que ajudam na elicitação e modelação de requisitos. Uma das abordagens de EROO bem estabelecidas é o KAOS. No entanto, em sistemas de larga escala, criar modelos KAOS pode resultar em modelos de objectivos incompletos e/ou complexos, o que pode dificultar a compreensão desses modelos bem como as suas modificações. Isto pode levar a um aumento dos custos de desenvolvimento do produto e da sua evolução. Assim, para sistemas de larga escala, a gestão eficaz da complexidade e completude dos modelos de objectivos é essencial.

Nesta dissertação de mestrado, propomos uma *framework* de métricas para apoiar a avaliação quantitativa da complexidade e completude dos modelos de objectivos KAOS. Estas métricas são especificadas formalmente, implementadas e incorporadas em uma ferramenta de modelação KAOS. Validamos então, as métricas através de um conjunto de casos de estudo reais e discutimos as práticas identificadas de modelação mais recorrentes.

Palavras-chave: Engenharia de Requisitos Orientada a Objectivos, Métricas de Complexidade de requisitos, Métricas de Completude de requisitos.

Abstract

Goal-Oriented Requirements Engineering (GORE) approaches have been developed to facilitate the requirements engineers work by, for example, providing abstraction mechanisms (i.e., the system's goals) to help eliciting and modeling requirements. One of the well-established GORE approaches is KAOS. Nevertheless, in large-scale systems building KAOS models may result in incomplete and/or complex goal models, which are difficult to understand and change. This may lead to an increase in costs of product development and evolution. Thus, for large-scale systems, the effective management of complexity and completeness of goal models is vital.

In this master's dissertation, we propose a metrics framework for supporting the quantitative assessment of complexity and completeness of KAOS goal models. Those metrics are formally specified, implemented and incorporated in a KAOS modeling tool. Then the metrics are validated with a set of real-world case studies and we discuss the identified recurring modeling practices.

Keywords: Goal-Oriented Requirements Engineering, Requirements Complexity Metrics, Requirements Completeness Metrics.

Conteúdo

1 Introdução 1
1.1 Motivação 1
1.2 Descrição e contexto 1
1.3 Objectivo do trabalho 2
1.4 Principais contribuições 3
1.5 Estrutura do documento 3
2 Enquadramento 5
2.1 Engenharia de Software 5
2.2 Engenharia de Requisitos 6
2.2.1 Categorias de requisitos 9
2.2.2 Problemas/Dificuldades 11
2.2.3 Tipos de abordagens de Requisitos 12
2.3 Metodologias de Engenharia de Requisitos Orientada a Objectivos 14
2.3.1 KAOS 14
2.3.2 NFR Framework 18
2.3.3 i* Framework 19
2.4 Métodos de controlo de qualidade de software 21
2.4.1 Abordagem GQM 21
2.4.2 Abordagem GSN 23
2.5 Sumário 25
3 Trabalho relacionado 27
3.1 AIRDoc 27
3.1.1 Passo1 – Elaboração do plano 28
3.1.2 Passo 2 – Definição do modelo GQM 29

3.1.3 Passo 3 – Recolha de dados 30
3.1.4 Passo 4 – Interpretação do modelo GQM 30
3.1.5 Passo 5 – Plano de melhoria do modelo de requisitos 31
3.1.6 Passo 6 – Requisitos de melhoria do modelo 31
3.2 Métricas *i para a integração da EROO com processos MDD 32**
3.2.1 Passo 1 – Formulação de métricas 32
3.2.2 Passo 2 – Declaração do metamodelo *i** 34
3.2.3 Passo 3 – Definição do modelo de validação *i** 35
3.2.4 Passo 4 – Especificação de métricas *i** 35
3.2.5 Passo 5 – Geração de extensões *i** 36
3.3 *i*MDF$_M$ 36
3.3.1 Passo 1 – Análise do domínio 37
3.3.2 Passo 2 – Análise das métricas do domínio 38
3.3.3 Passo 3 – Formulação de métricas *i** 38
3.3.4 Passo 4 – Actualizar a ferramenta *i*MDF 39
3.4 Análise das abordagens 39
3.5 Sumário 41

4 Métricas para a avaliação de modelos de objectivos KAOS 43
4.1 Introdução ao conjunto de métricas 43
4.2 modularKAOS 46
4.2.1 Introdução às técnologias usadas 47
4.2.2 Modificações sobre a ferramenta modularKAOS 48
4.2.3 Interpretador modularKAOS 50
4.3 Definição de métricas 51
4.3.1 Métricas para completude 55
4.3.2 Métricas para complexidade 59
4.4 Exemplo 69
4.5 Sumário 73

5 Validação 75
5.1 Casos de estudo 75
5.1.1 Sistema *Bay Area Rapid Transit* 75
5.1.2 Sistema *London Ambulance Service* 76
5.1.3 Sistema *Elevator* 76
5.1.4 Sistema *Meeting Scheduler* 76
5.1.5 Sistema *Library Management* 76
5.1.6 Sistema *SMART Home* 77
5.1.7 Sistema *Mine Safety Control* 77
5.1.8 Sistema *Car Park Management* 77

5.2 Resultados das métricas relacionadas com o objectivo de completude....78
5.3 Resultados das métricas relacionadas com o objectivo de complexidade....83
5.4 Sumário....87
6 Conclusão....89
6.1 Resumo....89
6.2 Limitações....90
6.3 Trabalho futuro....91
Bibliografia....93
A Metamodelo do modularKAOS....99
B Nova versão do metamodelo do modularKAOS....105
C modularKAOS - Manual do utilizador....109
C.1 Requisitos do sistema....109
C.2 Instalar e executar o modularKAOS....110
C.3 Criação de um projecto modularKAOS....110
C.4 Como efectuar modificações no modularKAOS?....110
D Métricas auxiliares....113
E Aplicação do modularKAOS aos casos de estudo....121

Lista de Figuras

FIGURA 2.1: CICLO DE VIDA DE UM SISTEMA DE *SOFTWARE* SEGUNDO O MODELO EM ESPIRAL [14] 6
FIGURA 2.2: CLASSES DE REQUISITOS NÃO-FUNCIONAIS [13] 10
FIGURA 2.3: CLASSIFICAÇÃO DOS REQUISITOS NÃO-FUNCIONAIS [17] 11
FIGURA 2.4: MODELOS DO MÉTODO KAOS [27] 16
FIGURA 2.5: EXEMPLO DE UM MODELO KAOS [28] 17
FIGURA 2.6: COMPONENTES DA FRAMEWORK NFR 19
FIGURA 2.7: PRINCIPAIS ELEMENTOS DA FRAMEWORK I* 21
FIGURA 2.8: ESTRUTURA DO MODELO GQM [34, 35] 22
FIGURA 2.9: PRINCIPAIS ELEMENTOS DA NOTAÇÃO GSN [38] 24
FIGURA 2.10: EXEMPLO DA APLICABILIDADE DA NOTAÇÃO GSN [38] 25
FIGURA 3.1: DIAGRAMA DE ACTIVIDADES DO AIRDOC [39] 28
FIGURA 3.2. PROCESSO DE INTEGRAÇÃO DE MÉTRICAS *I** [41] 32
FIGURA 3.3: MODELO DE VALIDAÇÃO [41] 35
FIGURA 3.4: DIAGRAMA DE ACTIVIDADES DO MÉTODO $IMDF_M$ [47] 37
FIGURA 3.5: EXEMPLO DO MAPEAMENTO DE UM DIAGRAMA ARQUITECTURAL EM BLOCOS [47] 38
FIGURA 4.1: FLUXO DE DESENVOLVIMENTO DO EDITOR GRÁFICO 47
FIGURA 4.2: ALTERAÇÕES NA CLASSE *KAOS* 48
FIGURA 4.3: NOVA CLASSE *PERFORMSLINK* 48
FIGURA 4.4: ALTERAÇÕES NAS CLASSES *REQUIREMENT* E *EXPECTATIONS* 49
FIGURA 4.5: ALTERAÇÕES NA CLASSE *OBSTACLE* 49
FIGURA 4.6: EXCERTO DO METAMODELO DA FERRAMENTA MODULARKAOS 53
FIGURA 4.7: DIAGRAMA GQM PARA OS MODELOS DE OBJECTIVOS KAOS 67
FIGURA 4.8: APLICAÇÃO DA FERRAMENTA MODULARKAOS E DAS MÉTRICAS AO CASO DE ESTUDO BART 71
FIGURA 5.1: PERCENTAGEM DE OBJECTIVOS FOLHA COM AGENTES 79
FIGURA 5.2: PERCENTAGEM DE OBJECTIVOS FOLHA COM OBJECTOS 80
FIGURA 5.3: PERCENTAGEM DE OBSTÁCULOS FOLHA COM RESOLUÇÕES 81
FIGURA 5.4: PERCENTAGEM DE OBJECTIVOS FOLHA COM OPERAÇÕES 82
FIGURA 5.5: PERCENTAGEM DE OPERAÇÕES COM AGENTES 83
FIGURA 5.6: NÚMERO DE OBJECTIVOS FOLHA POR AGENTE 84
FIGURA 5.7: OBJECTOS POR OBJECTIVO 85
FIGURA 5.8: ALTURA DO MODELO 86

FIGURA 5.9: NÚMERO DE SUB-OBJECTIVOS NO MODELO 87
FIGURA A.1: METAMODELO DO MODULARKAOS (VERSÃO RUI MONTEIRO) [12] 103
FIGURA B.1: METAMODELO DO MODULARKAOS (NOVA VERSÃO) 107
FIGURA C.1 PLUGINS NECESSÁRIOS PARA A UTILIZAÇÃO DO MODULARKAOS 109
FIGURA C.2: ELEMENTOS DO PROJECTO MODULARKAOS 111
FIGURA E.1: SISTEMA BARTS MODELADO PELA FERRAMENTA MODULARKAOS 123
FIGURA E.2: MÉTRICAS E AVISOS DO MODELO DO SISTEMA BARTS 125
FIGURA E.3: SISTEMA LASS MODELADO PELA FERRAMENTA MODULARKAOS 127
FIGURA E.4: MÉTRICAS E AVISOS DO MODELO DO SISTEMA LASS 129
FIGURA E.5: SISTEMA ES MODELADO PELA FERRAMENTA MODULARKAOS 131
FIGURA E.6: MÉTRICAS E AVISOS DO MODELO DO SISTEMA ES 133
FIGURA E.7: SISTEMA MSS MODELADO PELA FERRAMENTA MODULARKAOS 135
FIGURA E.8: MÉTRICAS E AVISOS DO MODELO DO SISTEMA MSS 137
FIGURA E.9: SISTEMA LMS MODELADO PELA FERRAMENTA MODULARKAOS 139
FIGURA E.10: MÉTRICAS E AVISOS DO MODELO DO SISTEMA LMS 141
FIGURA E.11: SISTEMA SHS MODELADO PELA FERRAMENTA MODULARKAOS 143
FIGURA E.12: MÉTRICAS E AVISOS DO MODELO DO SISTEMA SHS 145
FIGURA E.13: SISTEMA MSCS MODELADO PELA FERRAMENTA MODULARKAOS 147
FIGURA E.14: MÉTRICAS E AVISOS DO MODELO DO SISTEMA MSCS 149

Lista de Tabelas

TABELA 3.1: *TEMPLATE* PARA A DEFINIÇÃO DOS REQUISITOS DE QUALIDADE [39] 28
TABELA 3.2: *TEMPLATE* PARA A DEFINIÇÃO DAS QUESTÕES (PARCIAL) [39] 29
TABELA 3.3: *TEMPLATE* PARA A DEFINIÇÃO DE MÉTRICAS [39] 29
TABELA 3.4: *TEMPLATE* PARA A DEFINIÇÃO DE FUNÇÕES E HIPÓTESES QUE SUPORTAM A QUESTÃO Q1 [39] 30
TABELA 3.5: *TEMPLATE* PARA A APRESENTAÇÃO DOS VALORES DAS MÉTRICAS M1, M2 E M3 [39] 30
TABELA 3.6: *TEMPLATE* DE ANÁLISE DAS HIPÓTESES H1A E H1B [39] 31
TABELA 3.7: GUIA PARA A TRANSFORMAÇÃO DE MODELOS *I** PARA MODELOS DE CLASSES [41] 32
TABELA 3.8: APLICAÇÃO DO GQM PARA DEFINIR AS MÉTRICAS [41] 33
TABELA 3.9: EXEMPLO DA ESPECIFICAÇÃO DE MÉTRICAS COM A LINGUAGEM OCL [41] 36
TABELA 3.10: ANÁLISE DAS ABORDAGENS ESTUDADAS 39
TABELA 4.1: GQM PARA A AVALIAÇÃO DA COMPLETUDE 44
TABELA 4.2: GQM PARA A AVALIAÇÃO DA COMPLEXIDADE 44
TABELA 4.3: MÉTRICAS QUE SATISFAZEM O OBJECTIVO DE COMPLETUDE DA PERGUNTA P1 55
TABELA 4.4: MÉTRICAS QUE SATISFAZEM O OBJECTIVO DE COMPLETUDE DA PERGUNTA P2 56
TABELA 4.5: MÉTRICAS QUE SATISFAZEM O OBJECTIVO DE COMPLETUDE DA PERGUNTA P3 57
TABELA 4.6: MÉTRICAS QUE SATISFAZEM O OBJECTIVO DE COMPLETUDE DA PERGUNTA P4 58
TABELA 4.7: MÉTRICAS QUE SATISFAZEM O OBJECTIVO DE COMPLETUDE DA PERGUNTA P5 58
TABELA 4.8: MÉTRICAS QUE SATISFAZEM O OBJECTIVO DE COMPLEXIDADE DA PERGUNTA P6 59
TABELA 4.9: MÉTRICAS QUE SATISFAZEM O OBJECTIVO DE COMPLEXIDADE DA PERGUNTA P7 61
TABELA 4.10: MÉTRICAS QUE SATISFAZEM O OBJECTIVO DE COMPLEXIDADE DA PERGUNTA P8 63
TABELA 4.11: MÉTRICAS QUE SATISFAZEM O OBJECTIVO DE COMPLEXIDADE DA PERGUNTA P9 65
TABELA D.1: MÉTRICAS AUXILIARES 113

Lista de Acrónimos

AIRDoc	*Approach to Improve Requirements Documents*
DSL	*Domain Specific Language*
EMF	*Eclipse Modeling Framework*
EOL	*Epsilon Object Language*
ER	Engenharia de Requisitos
EROO	Engenharia de Requisitos Orientada a Objectivos
ES	Engenharia de *Software*
EVL	*Epsilon Validation Language*
GMF	*Graphical Modeling Framework*
GORE	*Goal-Oriented Requirements Engineering*
GQM	*Goal-Question-Metric*
GSN	*Goal Structuring Notation*
IEEE	*Institute of Electrical and Electronics Engineers*
*i*MDF$_M$	*i* Metrics Definition Framework Method*
KAOS	*Knowledge Acquisition in autOmated Specification*
LDE	Linguagens para Domínios Específicos
MDD	*Model-Driven Development*
NFR	*Non-Functional Requirements Framework*

OCL	*Object Constraint Language*
RE	*Requirements Engineering*
SD	*Strategic Dependency Model*
SE	*Software Engineering*
SIG	*Softgoal Interdependency Graph*
SR	*Strategic Rational Model*
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*

1

Introdução

1.1 Motivação

Actualmente, na comunidade de desenvolvimento de *software*, a utilização de métodos e linguagens de análise orientada a objectivos como o KAOS [1, 2], *i** [3, 4] ou NFR [5], têm o propósito de refinar e decompor as necessidades dos clientes em objectivos concretos durante a fase de elicitação de requisitos, que se encontra na fase inicial do desenvolvimento de *software*. Estes modelos visam representar organizações, especificando todas as partes envolvidas no *software* e todas as suas dependências que permitam atingir os objectivos. Uma vez construídos, estes modelos são utilizados para diferentes fins, sendo os mais importantes, a análise das propriedades que apresentam e a comparação entre alternativas, ou seja, entre as diferentes formas de representar os requisitos do sistema.

Tendo em conta que a construção destes modelos tem como propósito, na maioria das vezes, representar sistemas de grande escala, dificuldades na sua gestão e compreensão pode ter como consequências o aumento dos custos do desenvolvimento do produto, erros e inconsistências nas fases de desenvolvimento do produto posteriores, e um grande desagrado por parte dos clientes. A qualidade destes modelos deve então ser uma preocupação constante, e por esse motivo é uma área em aberto.

1.2 Descrição e contexto

Em [6], a Engenharia de *Software* aparece definida como "*a aplicação de uma abordagem sistemática, disciplinada, e quantificável, para o desenvolvimento, operação, e manutenção de softwares; ou seja, a aplicação de engenharia aos softwares*". Em

suma, a Engenharia de *Software* é uma área dedicada à criação e manutenção de sistemas de *software*, tendo em conta as preferências dos clientes. Esta área está dividida em várias áreas de conhecimento: requisitos, desenho, construção, teste, manutenção, gestão de configurações, gestão de engenharia, processos de engenharia, métodos e ferramentas de engenharia, e qualidade.

O foco deste trabalho é na área de Requisitos de *Software*. Os requisitos são definidos como propriedades que devem ser especificadas de forma a resolver algum problema do mundo real. A Engenharia de Requisitos inclui: elicitação de requisitos, especificação de requisitos, validação de requisitos, e gestão de requisitos [7]. Após a análise inicial, os requisitos do sistema são documentados. Existem vários paradigmas utilizados para a estruturação e representação de requisitos: objectivos, objectos, agentes, pontos de vista, cenários, e aspectos.

Esta dissertação estará centrada na especificação de requisitos orientada a objectivos, que é actualmente suportada por várias metodologias, sendo as principais o KAOS [1, 2], a *framework i** [3, 4], e o NFR [5]. Estas metodologias ajudam o analista a decompor os requisitos com base em objectivos, para melhor compreender o sistema.

1.3 Objectivo do trabalho

Dado o actual reconhecimento dos modelos de requisitos orientados a objectivos no processo de Engenharia de Requisitos, por exemplo ao nível da modelação e especificação de objectivos, e gestão de conflitos [8], o objectivo desta dissertação é encontrar uma forma de avaliar os modelos KAOS. Para esse fim, iremos propor uma abordagem quantitativa para a avaliação da complexidade e da completude destes modelos, com vista à identificação de oportunidades para a sua melhoria.

Ao longo desta dissertação usámos técnicas de Engenharia de *Software* Experimental para identificar oportunidades de melhorar a qualidade de modelos de requisitos orientados a objectivos. Identificámos oportunidades de investigação interessantes na avaliação de modelos KAOS, cobrindo assim uma metodologia ainda pouco explorada na vertente da avaliação de modelos, em contraste com o que acontece com o *i**, o qual tem sido alvo de maior atenção neste assunto.

1.4 Principais contribuições

De modo a suportar a avaliação quantitativa de modelos de requisitos em KAOS, pretendemos contribuir ao nível da definição, recolha e avaliação de métricas para modelos de requisitos.

As métricas serão propostas seguindo a abordagem *Goal-Question-Metric* (GQM) [9]. O GQM é uma técnica de definição de métricas muito utilizada pela comunidade de Engenharia de *Software* Experimental e recomendada pela própria *IEEE Computer Society* [10] como uma abordagem adequada à proposta de métricas. No Capítulo 4, pode ser vista a utilização desta abordagem na definição de um conjunto de métricas para a avaliação da complexidade e completude de modelos KAOS.

Outra contribuição é a extensão da ferramenta modularKAOS [11, 12] de modelação de modelos de objectivos KAOS que, por ter os seus modelos descritos em XML, permite automatizar o processo de recolha das métricas (ver Capítulo 4). Também auxilia no processo de avaliação dos modelos, por suportar a integração de métricas especificadas através da linguagem OCL.

Finalmente, a avaliação das métricas propostas também é um ponto a abordar nesta dissertação. Recorremos a uma avaliação experimental onde as métricas foram aplicadas a um conjunto de casos de estudo reais.

1.5 Estrutura do documento

Para além deste capítulo, de introdução, este documento encontra-se estruturado da seguinte forma:

- **Capítulo 2 – Enquadramento:** neste capítulo são apresentadas as bases desta dissertação, começando com a introdução à área da Engenharia de Software, passando pela fase de Elicitação de Requisitos, e terminando nas actuais metodologias que ajudam o processo de elicitação e de avaliação dos requisitos.
- **Capítulo 3 – Trabalho relacionado:** destacamos três abordagens em desenvolvimento, que estão relacionadas com o objectivo proposto neste documento, sendo elas: a metodologia AIRDoc, o processo de

definição de Métricas *i** para a integração da EROO com processos MDD, e a *framework*[1] *i*MDF$_M$.

- **Capítulo 4 – Métricas para a avaliação de modelos de objectivos KAOS:** neste capítulo começamos por descrever um conjunto de perguntas, através da aproximação GQM, com o objectivo de avaliar a completude e complexidade de modelos de objectivos KAOS. Depois introduzimos a ferramenta modularKAOS que irá ajudar na recolha e definição das métricas. Por fim, as métricas serão declaradas, bem como um exemplo do funcionamento da ferramenta com a integração das métricas propostas.
- **Capítulo 5 – Validação:** começamos por introduzir os casos de estudo aos quais iremos aplicar as métricas. Depois procedemos à validação das métricas, onde os resultados serão apresentados e discutidos.
- **Capítulo 6 – Conclusão:** neste último capítulo serão tiradas as conclusões sobre esta dissertação de mestrado e descritos alguns trabalhos futuros.

[1] Abstracção que une códigos comuns entre vários projectos de software provendo uma funcionalidade genérica.

2

Enquadramento

O objectivo deste capítulo é de introduzir o leitor às bases desta dissertação, nomeadamente ao trabalho relacionado com a Engenharia de Requisitos Orientada a Objectivos.

2.1 Engenharia de Software

E o que é Engenharia de *Software*? A Engenharia de *Software* vem exactamente trazer o conceito de engenharia para o desenvolvimento de sistemas de *software*, preocupando-se com todos os aspectos da sua produção [13]. Sommerville, em [14], definiu as seguintes actividades para o processo evolutivo de um *software*:

- **Identificação de requisitos:** onde são definidas as funcionalidades do *software* e as restrições a que este está sujeito.
- **Desenho:** onde é planeada uma solução para o problema, tendo em conta a fase anterior.
- **Implementação:** onde o desenho do *software* é reproduzido com o auxílio de uma linguagem de programação.
- **Validação:** onde o *software* é testado a fim de verificar se os requisitos estabelecidos pelo cliente foram cumpridos.
- **Evolução:** onde são corrigidos erros que apenas foram detectados após o *software* se tornar operacional. Nesta fase também é comum que o utilizador final apresente novos problemas a serem então implementados ou corrigidos.

O ciclo de vida de um sistema de *software* pode parecer um processo linear, mas é de facto um processo cíclico, onde cada fase pode interagir com as restantes. Existem vários modelos que definem a interacção entre as fases do processo evolutivo do *software*, por exemplo, Modelo em Cascata [15], Modelo em Espiral [16], Modelo de Desenvolvimento Evolutivo [13], entre outros. A Figura 2.1 mostra de uma forma geral o ciclo de vida de um sistema de *software*, tendo em conta a ideia do Modelo em Espiral, onde todas as fases são revisitadas até o *software* estar completo.

Figura 2.1: Ciclo de vida de um Sistema de *Software* segundo o Modelo em Espiral [14]

Face ao âmbito desta dissertação, a Engenharia de Requisitos irá ser estudada com mais pormenor na secção seguinte. Ao melhor entender esta fase, será mais fácil de compreender a importância das metodologias de requisitos orientados a objectivos que serão apresentados na Secção 2.3.

2.2 Engenharia de Requisitos

A Engenharia de Requisitos está relacionada com o início do ciclo de vida de um sistema de *software*. Esta etapa preocupa-se em identificar os problemas que precisam ser resolvidos para desenvolver um bom produto. Lamsweerde diz que "*…descobrir, formular, analisar e concordar em qual o problema a resolver, no porquê de o resolver, bem como no quem o deve resolver, são os objectivos da Engenharia de Requisitos*" [17]. Por outras palavras, Engenharia de Requisitos é a área que relaciona os objectivos e funcionalidades do sistema com os agentes e suas necessidades. Zave enfatiza a preocupação que a Engenharia de Requisitos tem em traduzir correctamente os objectivos do mundo real para o sistema de

software [18], e também a sua importância na evolução dos sistemas através da reutilização de especificações parciais [19].

Lamsweerde identifica três perguntas sobre o sistema, às quais os engenheiros devem ser capazes de responder [17]:

- **Porquê? –** que problema visa o sistema resolver;
- **O quê? –** quais as características desse sistema;
- **Quem? –** quem terá um papel activo sobre o sistema.

Existem diferentes processos que permitem responder a estas questões. Lamsweerde defende que a melhor forma é dada pela compilação das seguintes actividades [17, 20]:

- **Reconhecimento do domínio:** conhecer a estrutura organizacional e políticas da empresa, bem como, entrevistar e identificar *stakeholders*[2], são as grandes preocupações desta tarefa. Pretende-se com isto estudar o ambiente onde o *software* será inserido, e estimar as características, objectivos e problemas mais gerais.
- **Elicitação de requisitos:** através de uma colaboração entre *stakeholders* e engenheiros, identificar os requisitos e pressupostos para o sistema de *software*; de um conjunto de requisitos pobres resulta em um *software* pobre. Devido ao carácter crítico desta fase, espera-se que seja uma actividade feita minuciosamente.
- **Avaliação e concordância:** nesta fase realiza-se uma revisão independente dos materiais produzidos na fase anterior. Prevê-se que no final desta etapa todos os conflitos e riscos tenham sido resolvidos, e ambos *stakeholders* e engenheiros estejam satisfeitos e de acordo com o resultado final dos requisitos.
- **Especificação e documentação:** após estudar o domínio, entrevistar *stakeholders*, e identificar os requisitos e objectivos do sistema, espera-se que toda a informação recolhida nas etapas anteriores seja estruturada, documentada, e registada de forma persistente. O resultado é a criação de um documento de requisitos. Este

[2] Parte interessada ou interveniente (por exemplo, pessoa, organização, ou sistema), a quem as alterações feitas no sistema de *software* podem afectar directa ou indirectamente (por exemplo, clientes, investidores, accionistas, equipas de desenvolvimento, entre outros).

documento pode ser dividido em vários documentos de requisitos, específicos para cada *stakeholder*.

- **Consolidação de requisitos:** depois da criação do documento de requisitos, pretende-se assegurar a qualidade do que foi especificado. Para tal, espera-se que esta actividade venha corrigir possíveis inconsistências ou omissões, antes que o documento seja passado à equipa de desenvolvimento.

No entanto, Kotonya e Sommerville, em [21], apresentam um outro processo de Engenharia de Requisitos. As duas abordagens, de Kotonya e Sommerville e de Lamsweerde, apresentam uma linha de raciocínio muito semelhante. Kotonya e Sommerville apresentam cinco actividades:

- **Elicitação de requisitos:** esta actividade apresenta as mesmas características que as duas primeiras etapas de Lamsweerde, reconhecimento do domínio e elicitação de requisitos.
- **Análise e negociação de requisitos:** equivalente à actividade de análise e concordância apresentada por Lamsweerde.
- **Documentação de requisitos:** nesta fase pretende-se registar os requisitos anteriormente identificados – apresenta uma descrição equivalente à de Lamsweerde na fase de especificação e documentação.
- **Validação de requisitos:** esta fase é muito semelhante à última fase definida por Lamsweerde – consolidação de requisitos.
- **Gestão de requisitos:** o objectivo desta actividade consiste na gestão de mudanças dos requisitos acordados, na gestão do relacionamento entre requisitos, e na gestão de dependências entre o documento de requisitos e outros documentos produzidos durante o processo de engenharia de sistemas de *software*.

É apenas na última actividade defendida por Kotonya e Sommerville que se destaca a diferença entre estas duas aproximações. Estes autores preocupam-se com o facto de que, em qualquer fase do desenvolvimento de um sistema de *software*, possa ser necessário efectuar alterações aos requisitos estimados, ou até mesmo a necessidade de incluir novos requisitos. Sendo este tipo de situação comum, Kotonya e Sommerville incluíram uma actividade de gestão de requisitos.

Um outro processo da Engenharia de Requisitos é apresentado por Nuseibeh e Easterbrook, em [19], composto pelas seguintes actividades: elicitação de requisitos, modelação e análise de requisitos, comunicação de requisitos, concordância de requisitos, e evolução de requisitos. Esta abordagem assemelha-se muito à apresentada por Kotonya e Sommerville.

Para melhor entender o processo da Engenharia de Requisitos, nas seguintes secções serão abordados alguns conceitos de requisitos e revistos alguns dos problemas mais frequentes. Depois, apresentamos alguns paradigmas da Engenharia de Requisitos, que ajudam o processo de elicitação, e em especial o paradigma orientado a objectivos. Mais à frente introduzimos duas abordagens que ajudam a garantir a qualidade dos requisitos.

2.2.1 Categorias de requisitos

Nas secções anteriores falámos das áreas mais gerais. Explicámos a importância da Engenharia de Software, e as vantagens da Engenharia de Requisitos no desenvolvimento de sistemas de *software*. No entanto, ainda não foi dada uma definição mais detalhada do que são requisitos, neste contexto. Bahill e Dean, em [22], afirmam que um requisito *"é uma declaração que identifica um recurso ou função necessários por um sistema, a fim de satisfazer as necessidades dos seus clientes"*; portanto, quando falamos de requisitos, podemos nos estar a referir a um atributo, característica, ou qualidade que o sistema possui de forma a ter a utilidade desejada. Há que mencionar, que um requisito informa o que o sistema deve fazer, mas não como o fazer. Seguem-se alguns exemplos:

- *"Enquanto a máquina de café estiver no processo de aquecimento deve ter o botão de tirar café intermitente."*
- *"A porta de um elevador deve manter-se fechada enquanto este se move."*
- *"Um cliente de um ginásio só pode assistir a uma aula em simultâneo."*

Quando falamos em requisitos no contexto de Engenharia de Requisitos, um requisito representa o que o sistema deve ser capaz de fazer (ou seja, responder à pergunta *O quê?*) – requisito funcional – ou o quão bem é capaz de realizar determinada função – requisito não-funcional.

Requisitos funcionais, em Engenharia de Requisitos, representam os serviços que o sistema terá de oferecer, de uma forma automática, ao utilizador final, podendo explicitar condições relativamente ao meio onde opera – comportamento específico do sistema. Por exemplo:

- *"O sistema para a gestão de um ginásio deve ser capaz de gerir as lotações das aulas."*
- *"As portas de um comboio apenas podem ser abertas quando este pára."*

Os **requisitos não-funcionais** explicam como o sistema de *software* deve ser desenvolvido, indicando a forma como este satisfaz os requisitos funcionais e apresentando restrições às quais o sistema deve obedecer. Por exemplo:

- *"Quando um cliente de um ginásio passa o seu cartão para entrar no ginásio, as cancelas devem permitir a entrada num intervalo de tempo não superior a 2 segundos."*
- *"O sistema de informação do ginásio deve estar inacessível da meia-noite até a 1 hora da manhã para realizar cópias de segurança."*

Os requisitos não-funcionais são também conhecidos por atributos de qualidade [17]. Esses atributos podem ser: segurança, precisão, usabilidade, desempenho, sustentabilidade, entre muitos outros.

Sommerville [13] apresenta uma proposta para a classificação dos requisitos não-funcionais, onde indica que estes podem surgir de características do produto, da organização, e de fontes externas, como pode ser visto na Figura 2.2.

Figura 2.2: Classes de Requisitos Não-Funcionais [13]

Outra proposta aparece em [17] por Lamsweerde, onde os requisitos são classificados em qualidade do serviço, restrições de arquitectura e de desenvolvimento, e cumprimento de leis nacionais. A Figura 2.3 descreve esta decomposição.

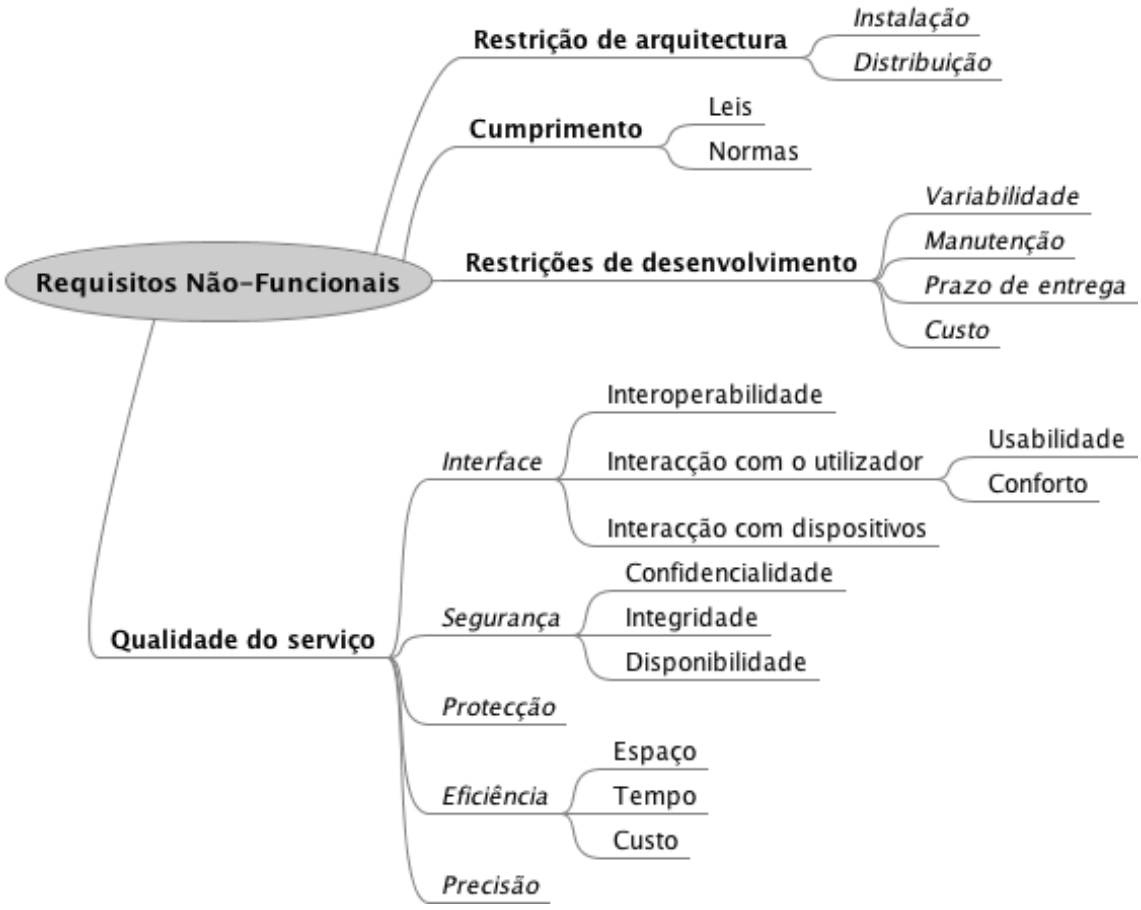


Figura 2.3: Classificação dos Requisitos Não-Funcionais [17]

A grande diferença entre a classificação de Lamsweerde (Figura 2.3) para a classificação de Sommerville (Figura 2.2) é o suporte que Lamsweerde dá a requisitos de desenvolvimento relacionados com a forma como o sistema de *software* deve ser desenvolvido. Quanto às restantes classificações: a classe de requisitos organizacionais de Sommerville (Figura 2.2) corresponde às classes de requisitos de arquitectura e de cumprimento de leis e normas de Lamsweerde (Figura 2.3); e a classe de qualidade do serviço de Lamsweerde (Figura 2.3) cobre a essência das classes de requisitos do produto e de requisitos externos de Sommerville (Figura 2.2).

2.2.2 Problemas/Dificuldades

A Engenharia de Requisitos, como já foi referido, é a primeira fase do processo de desenvolvimento de *software*, e por esse motivo é natural que aí surjam problemas que, se não resolvidos, irão prejudicar as seguintes fases.

A primeira dificuldade é o facto de os *softwares* apresentarem particularidades, mesmo que pequenas, tornando muito difícil automatizar

todo o processo da Engenharia de Requisitos. Posto isto, é preciso que os interessados no *software* (ou seja, os seus *stakeholders*) resolvam o problema, podendo ser eles próprios o segundo grande problema, muitas vezes por erros de comunicação e falha na interpretação dos requisitos do cliente.

Por exemplo, uma falha comum é se houver pouco cuidado ao documentar os requisitos, pode ocorrer erros de omissão de requisitos, contradição entre eles, situações de ambiguidade, falsos pressupostos, entre outros. Outra situação de graves resultados é quando há falta de cuidado face aos requisitos não-funcionais. Por não serem tão óbvios como os requisitos funcionais por vezes são negligenciados ou endereçados apenas em fases posteriores do desenvolvimento, mas há que ter em conta que os atributos de qualidade são tão ou mais importantes que os requisitos funcionais. Estes problemas agravam-se no desenvolvimento de sistemas de larga escala, onde a sua complexidade pode comprometer a completude dos requisitos.

Portanto, o objectivo desta dissertação está relacionado com a complexidade e a completude dos modelos de requisitos, visto que a complexidade de modelos representativos dos requisitos de um sistema não ajuda em nada na sua compreensão. Como tal, pretende-se minimizar este factor sem afectar a sua completude.

Todos estes erros acontecem por falha humana, e uma forma de os prevenir é realizando todos os passos do processo da Engenharia de Software cuidadosamente, e revisitar estas etapas sempre que necessário. Não nos podemos esquecer que os erros que ocorrem na Engenharia de Requisitos e que são detectados numa fase de desenvolvimento posterior são os mais caros de resolver, resultando muitas vezes em restruturações prejudiciais ao cliente, e também os mais perigosos, devido ao carácter crítico de muitos sistemas de *software*, por exemplo, Sistema de uma Estação Nuclear, Sistema de Controlo de um Avião, entre outros [17].

Para prevenir estes erros têm vindo a ser desenvolvidas técnicas para melhorar a comunicação entre as partes envolvidas. Na Secção 2.3 serão apresentadas algumas dessas metodologias.

2.2.3 Tipos de abordagens de Requisitos

Em Engenharia de Requisitos existem várias técnicas utilizadas para descrever requisitos de um sistema de *software*. Segue-se a introdução a algumas dessas técnicas.

Engenharia de Requisitos Orientada a *Viewpoints*. Este tipo de abordagem consiste na recolha de vários *viewpoints*. Estes representam um conjunto de informação sobre o sistema do ponto de vista de um *stakeholder* [21]. Ao armazenar pontos de vista de diferentes fontes, torna-se mais fácil perceber os possíveis conflitos de ideias entre *stakeholders*, garantindo uma melhor compreensão sobre o sistema de *software* a desenvolver [13].

Engenharia de Requisitos Orientada a Cenários [13]. Este tipo de abordagem utiliza exemplos de comportamentos de sistemas existentes – cenários – para completar a descrição dos requisitos do sistema de *software* a desenvolver. Tendo em conta que as pessoas se costumam identificar com mais facilidade a exemplos reais, esta técnica joga a favor disso, facilitando, através de exemplos, a compreensão por parte dos *stakeholders*.

Engenharia de Requisitos Orientada a Aspectos [23]. Esta abordagem foca-se na identificação, definição e representação de propriedades transversais[3] (por exemplo, segurança, mobilidade, disponibilidade, entre outras) ao nível do requisito. Desta forma é possível capturar a influência destas propriedades noutros requisitos do sistema, reforçando a modularidade desse sistema.

Engenharia de Requisitos Orientada a Objectos [24]. Aqui, os requisitos do sistema de *software* são traduzidos por meio de objectos, que contêm informação acerca da sua funcionalidade, do seu comportamento, e das interacções com outros objectos. O UML [25], acrónimo para *Unified Modeling Language*, é a notação mais conhecida e utilizada que segue este tipo de abordagem.

Engenharia de Requisitos Orientada a Objectivos [8]. Objectivos são declarações que exprimem propriedades, funcionais ou não-funcionais, que o sistema deve garantir. Este tipo de abordagem utiliza o conceito de objectivo para estruturar os requisitos do sistema. O grande interesse desta abordagem é o facto de os objectivos poderem ser formulados em diferentes níveis de abstracção, tornando a explicação dos requisitos mais intuitiva para os *stakeholders*.

Na próxima secção serão apresentadas as principais metodologias face à técnica de Engenharia de Requisitos Orientada a Objectivos, nomeadamente, KAOS, NFR e *i**.

[3] Propriedade de um programa que afecta outras partes de um sistema, em que a sua funcionalidade encontra-se dispersa por vários módulos desse sistema.

2.3 Metodologias de Engenharia de Requisitos Orientada a Objectivos

Em Engenharia de Requisitos Orientada a Objectivos os requisitos são apresentados por forma de objectivos que o sistema deve ser capaz de garantir. Para tal, essas exigências são identificadas, analisadas e atribuídas a componentes do sistema ou a agentes do ambiente. As principais motivações desta visão da Engenharia de Requisitos Orientada a Objectivos são [8]:

- **Comunicação e compreensão** – devido aos vários níveis de abstracção obtidos pelo refinamento dos objectivos, esta área fornece um quadro global para documentar os requisitos, e também uma forma eficaz de comunicar com os *stakeholders*;
- **Variabilidade** – através do refinamento alternativo dos objectivos e da atribuição alternativa de responsabilidades, é possível explorar várias configurações do sistema;
- **Completude –** a formalização dos objectivos permite provar a completude e a integridade do sistema;
- **Rastreabilidade –** o refinamento dos objectivos oferece rastreabilidade vertical dos objectivos de alto nível para os objectivos mais detalhados;
- **Gestão de conflitos –** os objectivos podem ser usados para detectar e gerir conflitos entre os requisitos.

Seguem-se as principais metodologias de Engenharia de Requisitos Orientada a Objectivos que visam oferecer estas características. Em [26] é feita uma análise e comparação entre estas três abordagens.

2.3.1 KAOS

KAOS [1, 2] (do inglês ***K****nowledge* ***A****cquisition in aut****O****mated* ***S****pecification*), é uma metodologia criada por Lamsweerde, que pretende dar suporte a todo o processo de elaboração e aquisição de requisitos, baseando-se na decomposição e refinamento de objectivos. Esta abordagem é apoiada pelos seguintes conceitos e terminologias principais: objectivos, agentes, objectos, operações, obstáculos, e conflitos.

Os **objectivos** são metas que o sistema deve ser capaz de cumprir. Estes podem ser decompostos e descritos por objectivos de baixo nível, e existem

duas formas para o fazer: *AND-refinement*, e *OR-refinement*. A primeira consiste em decompor o objectivo de alto-nível em vários objectivos intermédios, que no seu conjunto compõem o objectivo de alto-nível. A segunda consiste em apresentar formas alternativas que o sistema tem para garantir o mesmo objectivo. De forma a completar os modelos, espera-se que todo o refinamento termine com a atribuição desse objectivo a um agente e opcionalmente a um objecto. Os objectivos folha podem ser requisitos, quando o seu cumprimento depende de agentes do sistema, ou expectativas, quando associados a agentes do ambiente.

Os **agentes** são entidades responsáveis em actuar sobre determinadas operações ou objectivos. Podem ser agentes do ambiente (por exemplo, pessoas) ou agentes do sistema (por exemplo, programas).

Os **objectos**, ou entidades, são elementos de interesse no sistema caracterizados por um conjunto de atributos que definem diferentes estados nos quais o objecto pode transitar. Para tal existem as **operações**, que permitem essa transição dos objectos através de relações de entrada e saída com o objecto. As operações são caracterizadas por pré-condições, pós-condições, e condições de desencadeamento.

O KAOS preocupa-se com a identificação de inconsistências dentro do domínio do sistema. Estas inconsistências podem ser: **obstáculos**, quando os objectivos identificados podem estar inconsistentes com o domínio da aplicação; ou **conflitos**, quando os pontos de vista, depois de formalizados, dão origem a objectivos inconsistentes.

Os principais passos que esta abordagem oferece para a especificação de requisitos são [2, 27]:

- **Elaboração de objectivos:** refinamento de objectivos por meio da identificação de novos objectivos mais específicos que caracterizem os de alto nível.
- **Identificação de objectos:** identificar objectos na formulação do objectivo, definir a sua ligação, e descrever as propriedades do domínio.
- **Identificação de operações:** identificar as transições do estado do objecto que são significantes para o objectivo.

- **Operacionalização dos objectivos:** determinar as condições e invariantes das operações de forma a assegurar todos os objectivos.
- **Atribuição de responsabilidades:** identificar responsabilidades alternativas para os objectivos folha e atribuir operações a agentes.

A Figura 2.4 ajuda a compreender a relação entre os principais conceitos do KAOS, e sua integração com o tipo de modelos que esta abordagem permite construir.

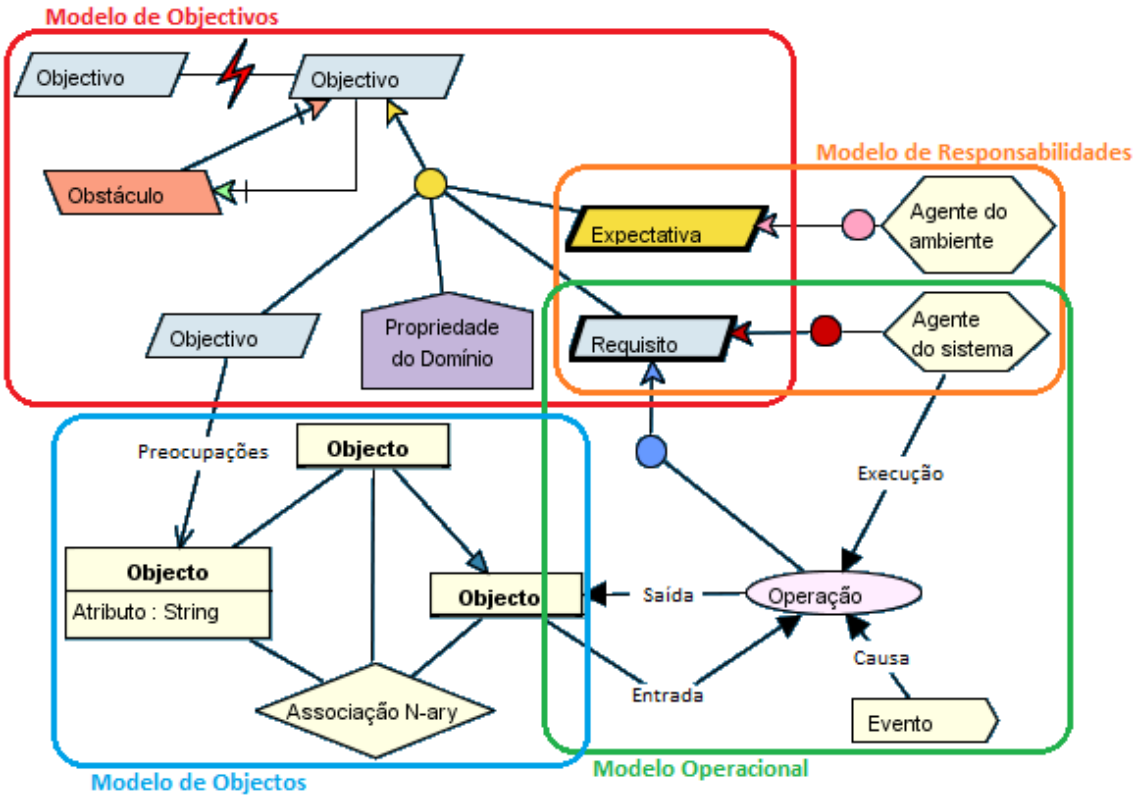


Figura 2.4: Modelos do método KAOS [27]

A Figura 2.5 apresenta um modelo concreto KAOS que modela a funcionalidade de controlo de acessos de um sistema de um ginásio [28]. Na figura podemos observar diferentes elementos desta metodologia: objectivos e seus refinamentos, agentes e suas ligações com requisitos e expectativas, e relações entre objectos e objectivos. Este exemplo é modelado com o auxílio da ferramenta Objectiver.

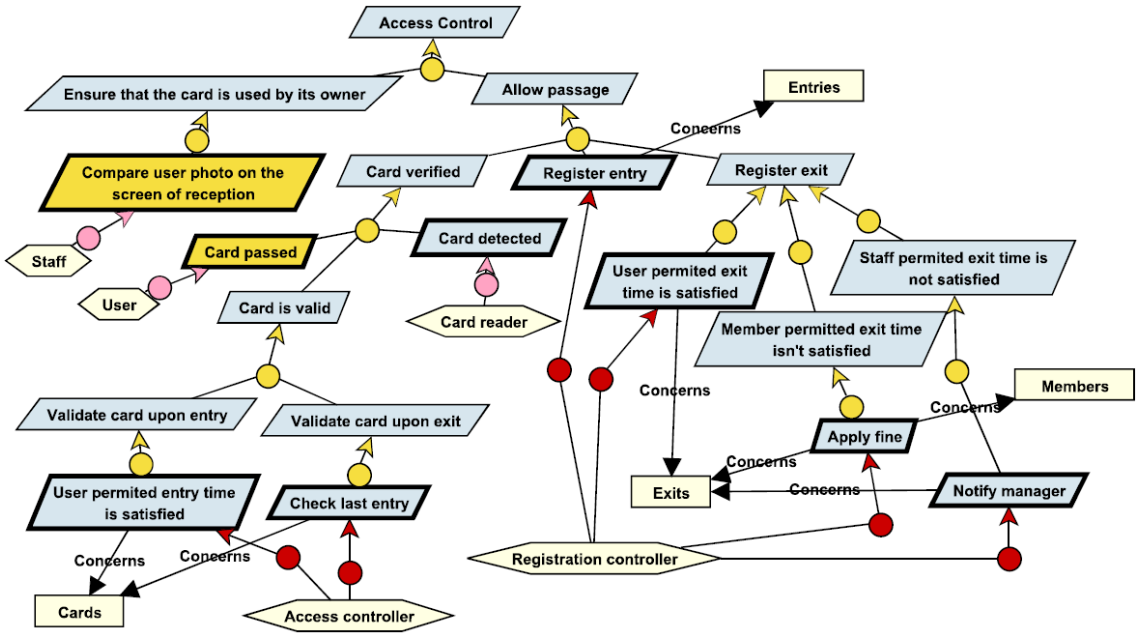


Figura 2.5: Exemplo de um modelo KAOS [28]

Os modelos KAOS podem ser definidos recorrendo a ferramentas como o Objectiver ou o modularKAOS.

O Objectiver [29] é uma ferramenta comercial de modelação e suporte para a escrita de documentos de requisitos baseada na metodologia de Engenharia de Requisitos, KAOS. As suas principais capacidades residem em: identificar requisitos; criar modelos KAOS (objectivos, responsabilidades, objectos e operações); e gerar documentos de requisitos.

O modularKAOS é uma ferramenta experimental desenvolvida no âmbito da dissertação de mestrado de Dias em [11], e posteriormente aperfeiçoada por Monteiro na sua dissertação de mestrado [12]. Trata-se de um editor de modelos KAOS baseado em Linguagens para Domínios Específicos, que tem como principal objectivo melhorar a escalabilidade dos modelos KAOS, face à ferramenta Objectiver, através da introdução de um novo conceito de Compartimento. Esta ferramenta permite a representação dos modelos de objectivos, responsabilidades e objectos, falhando apenas no suporte para os modelos de operações.

Ambas as ferramentas conseguem representar o modelo de objectivos do paradigma KAOS, e apesar de o Objectiver oferecer um maior conjunto de funcionalidades, falha naquele que nos é mais importante – o acesso a toda a informação contida nos ficheiros por um programa externo (visto ser um programa comercial todos os seus ficheiros encontram-se num formato

desconhecido). Perante este cenário, a utilização do modularKAOS torna-se bastante aliciante, pois este, ao ser baseado no paradigma das Linguagens para Domínios Específicos, oferece um maior nível de abstracção e controlo total sobre todos os ficheiros gerados. No Capítulo 4 iremos apresentar com mais detalhe esta ferramenta.

2.3.2 NFR Framework

Tal como o nome indica, o NFR [5], acrónimo para ***N**on-**F**unctional **R**equirements* (em português Requisitos Não-Funcionais), é uma estrutura de suporte à modelação e análise de requisitos não-funcionais. A sua estrutura é baseada em grafos SIG (**G**rafos de **I**nterdependência de ***S**oftgoals*) que descrevem as dependências entre objectivos e como estes são decompostos. O seu objectivo é ajudar os analistas a perceberem o impacto que as suas decisões terão perante os requisitos não-funcionais, tais como o desempenho, segurança, precisão, entre outros.

Um grafo é composto por nós e relações entre nós. Igualmente nesta abordagem os conceitos podem ser definidos entre nós e as suas relações, sendo os nós designados por *softgoals* NFR. Estes são objectivos que o sistema deve ser capaz de cumprir, e podem ser:

- **requisitos não-funcionais –** ver Secção 2.2.1;
- ***softgoals* de operacionalização** – que modelam técnicas para satisfazer os *softgoals* NFR;
- ***softgoals* de argumentação** – que permitem ao analista registar a lógica de desenho relativamente a objectivos e decomposições.

As relações entre nós podem ser feitas de duas formas:

- **interdependências** – conjunto de relacionamentos entre os *softgoals* (por exemplo, *And*, *Or*, *Equal*);
- **correlações** – conjunto de regras que permitem inferir interacções negativas ou positivas entre os *softgoals* (por exemplo, *Break*, *Make*, *Hurt*, *Help*, *Some+*, *Some-*).

Outro aspecto interessante na abordagem NFR *Framework* é o processo de avaliação dos *softgoals*, que mede o impacto que a importância de cada objectivo tem face aos objectivos de alto nível. Esta avaliação pode ser: *denied*; *weakly denied*; *undecided*; *conflict*; *satisfied*; *weakly satisfied*.

A Figura 2.6 mostra os principais componentes da metodologia NFR.

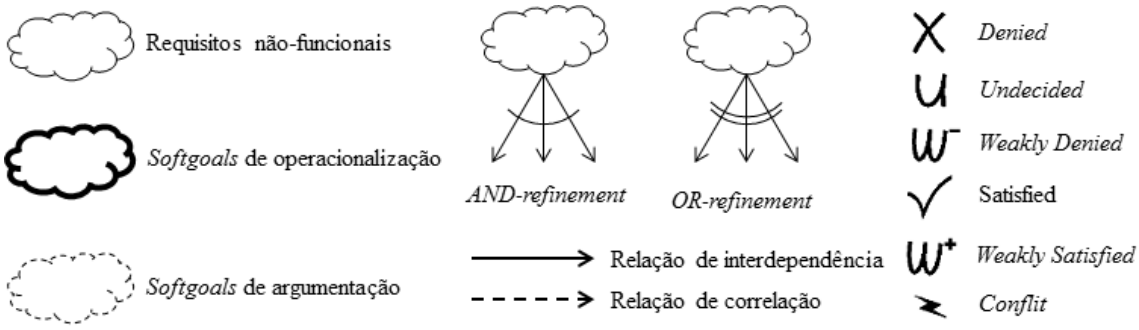


Figura 2.6: Componentes da framework NFR

A ferramenta NFR baseia-se de igual forma no refinamento dos objectivos não-funcionais. Através dos dois tipos de refinamentos, anteriormente visitados, *AND-refinement* e *OR-refinement*, é possível decompor os objectivos e também reconhecer várias formas de os garantir.

2.3.3 i* Framework

A *framework i** [3, 4] é uma ferramenta de modelação apropriada para modelar sistemas na fase de elicitação de requisitos de forma a melhor conhecer o domínio do problema. Esta ferramenta é orientada a agentes e a objectivos com o propósito de modelar ambientes organizacionais. Para tal, oferece uma representação formal das relações estratégicas entre agentes e seus comportamentos. Ou seja, o objectivo desta metodologia é analisar a forma como os objectivos dos diferentes actores podem ser alcançados, tendo em conta as relações entre actores humanos e o sistema. Os seus principais conceitos são:

- **actores** – abstracção utilizada para referir entidades activas (por exemplo, humanos, *hardware*, *software*) capazes de acções independentes;
- **objectivos** – que estão associados aos actores e são requisitos que devem ser cumpridos no sistema;
- **tarefas** – indicações sobre algo que deve ser feito pelos actores e podem ser vistas como uma solução no sistema (por exemplo, operações, processos, representações de dados, entre outras) que se pretende desenvolver;
- ***softgoals*** – requisitos não-funcionais que devem ser cumpridos pelo sistema;

- **recursos** – entidades, físicas ou não, que devem ser disponibilizadas ao sistema.

Para interligar os conceitos anteriores são utilizadas as seguintes ligações: dependência, *means-end*, decomposição e contribuição.

Esta *framework* oferece dois tipos de modelos relativos a diferentes níveis de abstracção: Modelo de Dependência Estratégicas (do inglês *Strategic Dependency Model*) e Modelo de Raciocínio Estratégico (do inglês *Strategic Rationale Model*).

O modelo de Dependência Estratégica é composto por um conjunto de relações de dependência entre os actores do sistema. Tem como objectivo captar a intenção dos processos dentro da organização e a importância que cada um desses processos tem face aos participantes, garantindo a abstracção quanto aos restantes detalhes do sistema. O interesse destes modelos reside exactamente nesse nível de abstracção, permitindo ao analista ter a percepção do impacto que a introdução de novos processos tem na organização.

A única relação permitida neste modelo é a de **dependência**. Esta indica que um actor (*depender*) depende de outro actor (*dependee*) para algo (*dependum*). O *dependum* pode ser um objectivo, uma tarefa, um recurso, ou um *softgoal*.

Os modelos de Raciocínio Estratégico são usados para explorar a lógica por detrás dos processos no sistema, descrevendo os interesses e preocupações do sistema. Este modelo é complementar ao modelo de Dependência Estratégica, especificando, para além dos actores e suas dependências, os objectivos funcionais e não funcionais, as tarefas e os recursos, utilizando as fronteiras de cada actor para especificar assuntos internos a esse actor.

As relações permitidas no modelo de Raciocínio Estratégico são:

- ***means-end*** – utilizado para ligar uma tarefa a um objectivo, indicando uma forma específica de atingir um objectivo;
- **decomposição** – capaz de especificar uma tarefa, indicando as sub-tarefas, sub-objectivos, recursos e *softgoals* necessários para satisfazer essa mesma tarefa;
- **contribuição** – serve para indicar como uma tarefa contribui para alcançar os atributos de qualidade especificados pelos *softgoals*.

A Figura 2.7 mostra as principais terminologias e conceitos que complementam os modelos *i**.

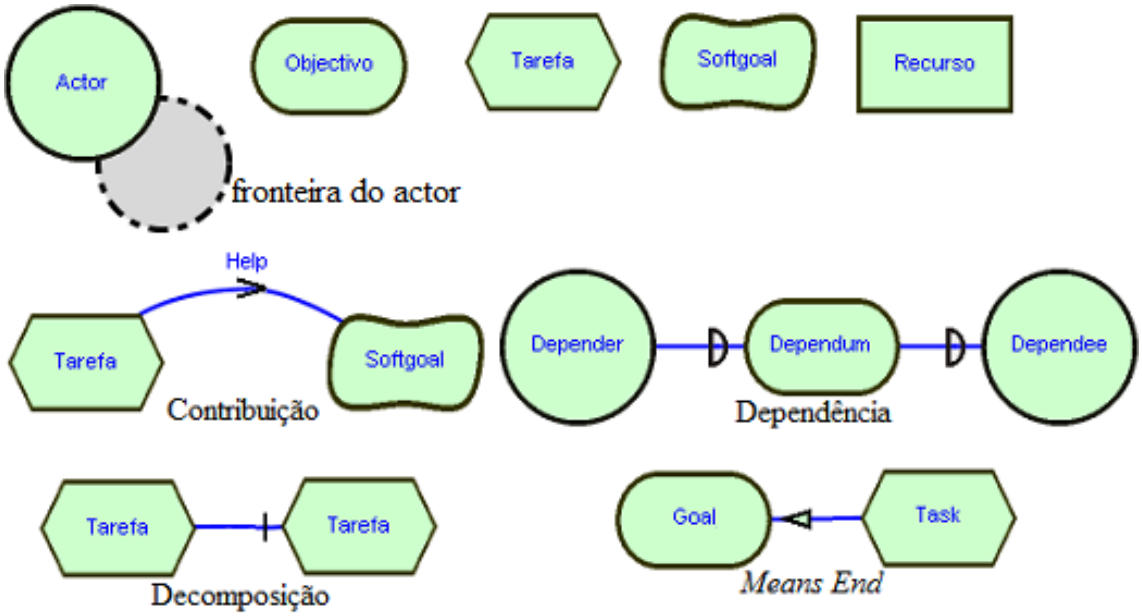


Figura 2.7: Principais elementos da framework i*

2.4 Métodos de controlo de qualidade de software

A necessidade de assegurar a qualidade dos produtos de *software* tem sido um requisito em crescimento, devido ao aumento da competitividade dos mercados. Para garantir a qualidade, várias iniciativas têm sido procuradas, tais como, *benchmarking*, medições, validação do sistema, entre outras [30, 31].

No âmbito desta dissertação, iremos apresentar duas aproximações focadas na medição e na validação de objectivos, GQM (*Goal-Question-Metric*) e GSN (*Goal Structuring Notation*), respectivamente. A primeira consiste em definir um conjunto de métricas nas quais as propriedades do sistema podem ser medidas. A segunda, mais focada para o desenho de aplicações de segurança crítica, consiste em garantir que as especificações do sistema estejam de acordo com as necessidades do utilizador.

2.4.1 Abordagem GQM

GQM, o acrónimo para ***G**oal-**Q**uestion-**M**etric* (em português Objectivo-Pergunta-Métrica), é uma metodologia de monitorização e medição do desempenho de actividades de *software* desenvolvida por Basili e pela sua equipa do *NASA Software Engineering Laboratory* [9]. Este paradigma é uma abordagem orientada a objectivos para medições de sistemas de desenvolvimento de *software* que prevê três níveis de abstracção:

- ***Nível 1.*** *Nível Conceptual (Objectivo)* – identificar objectivos de medição;
- ***Nível 2.*** *Nível Operacional (Questão)* – propor questões para os objectivos de medição;
- ***Nível 3.*** *Nível Quantitativo (Métrica)* – definir métricas que ajudem a responder às perguntas colocadas.

Com mais de duas de dezenas de prémios científicos, cerca de duas centenas de artigos publicados, e participação em vários livros, Basili é um dos pioneiros em estudos que usam a medição e avaliação como ferramentas para ajudar a melhorar o processo de desenvolvimento de *software* [32, 33]. Em [34], Basili descreve a utilização da aproximação GQM como: *"Escrever os objectivos permitiu que nos concentrássemos sobre as questões importantes. Definir questões permitiu-nos tornar os objectivos mais específicos e sugerir as métricas relevantes a esses objectivos. Esta estrutura permitiu-nos ver a relação completa entre os objectivos e métricas, determinar quais os objectivos e métricas que estavam em falta ou inconsistentes, e fornecer um contexto para a interpretação dos dados depois de terem sido recolhidos."*.

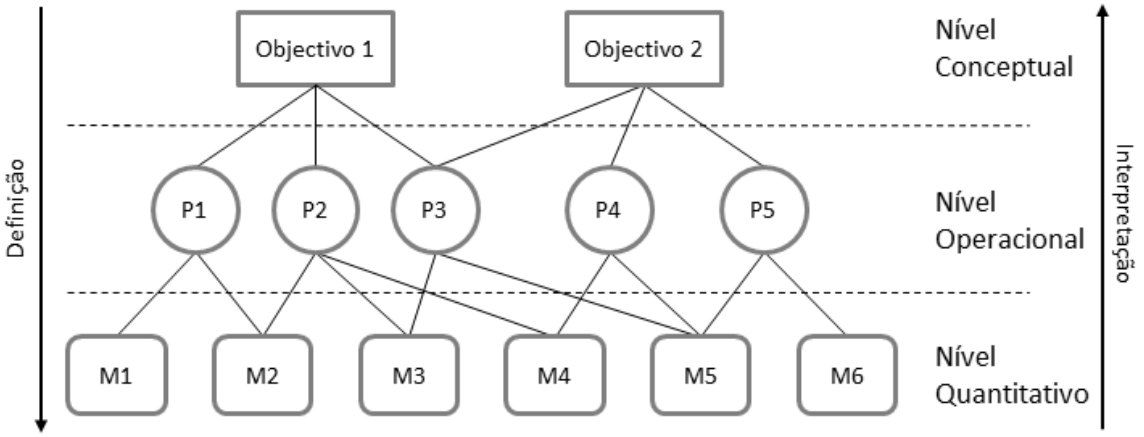


Figura 2.8: Estrutura do modelo GQM [34, 35]

O modelo GQM [34-36] (ilustrado na Figura 2.8) começa por identificar um objectivo de medição específico (nível conceptual), tendo em conta a entidade, propósito, atributos de qualidade, pontos de vista e ambiente. Os objectivos de medição podem ser de três tipos: produtos (por exemplo, programas, testes); processos (por exemplo, testar, desenhar, entrevistar); ou recursos (por exemplo, hardware, pessoal). Cada objectivo é então refinado em várias questões (nível operacional), que caracterizam a forma como um determinado objectivo poderá ser obtido. Depois, são identificadas as métricas

(nível quantitativo), que providenciam toda a informação quantitativa necessária para responder às questões do nível anterior. Esta informação pode ser objectiva, se depender apenas do objecto que está a ser medido (por exemplo, número de versões de um documento, tamanho do programa), ou subjectiva, quando depende do objecto e também do ponto de vista do *stakeholder* (por exemplo, nível de satisfação do utilizador, legibilidade do texto).

A possibilidade que a abordagem GQM fornece em definir questões abstractas e responder com o auxílio de métricas, a fim de avaliar a qualidade, traz um enorme interesse para o âmbito desta dissertação, que visa, exactamente, controlar alguns atributos de qualidade. Para além disto, é uma abordagem bem conceituada na comunidade, sendo aconselhada pelo próprio *Software Engineering Standards Committee* da *IEEE Computer Society* como uma *framework* adequada para estabelecer métricas ao nível organizacional [10]. Por estes motivos, a solução proposta por esta dissertação terá em consideração a abordagem GQM.

No Capítulo 4 será apresentado um exemplo da aplicação da abordagem GQM para um problema no âmbito desta dissertação.

2.4.2 Abordagem GSN

A abordagem GSN, o acrónimo para ***G**oal **S**tructuring **N**otation* (em português, Notação para a Estruturação de Objectivos), foi desenvolvida para descrever argumentos de segurança (tais como, requisitos, reivindicações, provas, e contextos), e também representar as relações entre os argumentos. Ou seja, o GSN pretende explicar como é que cada requisito pode ser suportado por exigências do sistema, e de que forma é que essas reivindicações são apoiadas por provas e pelo contexto definido para o argumento [37]. Para melhor entender a notação GSN devemos começar por introduzir a necessidade de argumentos de segurança.

As equipas de desenvolvimento começaram a ver a necessidade de provar que o nível de segurança dos sistemas de *software* era aceitável. Foi então criado o conceito de argumento de segurança (do inglês *safety case*), que através de evidências e argumentos conseguem garantir determinados requisitos e objectivos [37].

A notação de GSN surgiu para clarificar a forma como os argumentos de segurança são apresentados. GSN consiste nos seguintes termos [38]:

- **Objectivos:** requisitos, alvos ou restrições, que o sistema deve suportar;
- **Modelos:** um objectivo é expresso em termos de um modelo do sistema, que pode ser, por exemplo, um modelo arquitectural, ou uma descrição de um processo, entre outros;
- **Estratégias:** os objectivos são resolvidos por uma estratégia, que divide o objectivo em vários sub-objectivos;
- **Justificações:** as justificações são razões ou evidências que suportam a utilização de uma determinada estratégia;
- **Meta-estratégias:** trata-se de estratégias alternativas para uma atingir a solução de um ou mais objectivos;
- **Critérios:** servem para avaliar se um objectivo foi resolvido de forma satisfatória. Normalmente são atribuídas medidas e procedimentos para avaliar a satisfação desse objectivo;
- **Restrições:** servem para restringir a forma como um objectivo é resolvido;
- **Soluções:** as soluções são análises, evidências, resultados dos relatórios de auditoria, ou referências à concepção material.

Os principais símbolos da notação GSN são apresentados na Figura 2.9.

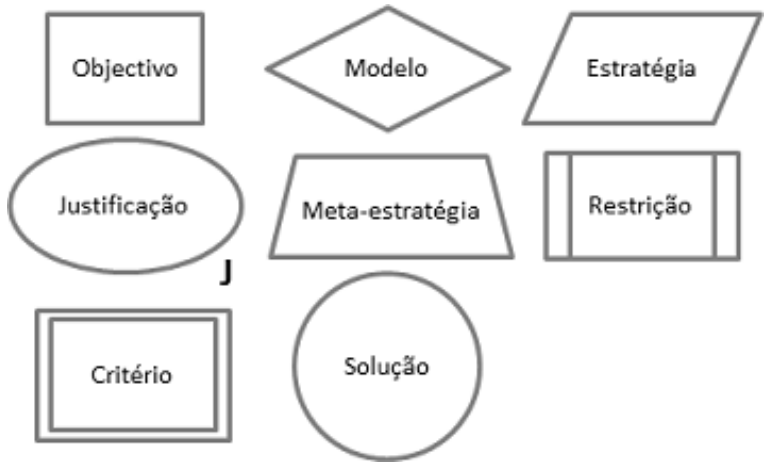


Figura 2.9: Principais elementos da notação GSN [38]

O principal propósito de qualquer estrutura de objectivos, composta pelos elementos referidos na Figura 2.9, é mostrar como requisitos do sistema podem ser divididos sucessivamente em novos objectivos por referência directa a novas evidências [37].

Apresenta-se na Figura 2.10 um exemplo da aplicação do GSN. A hierarquia apresentada descreve o argumento de segurança para prevenir falhas das válvulas (designadas *Gag Failures*) do sistema de um reactor nuclear (designado *Advanced Gas Reactor Nuclear Trip System*).

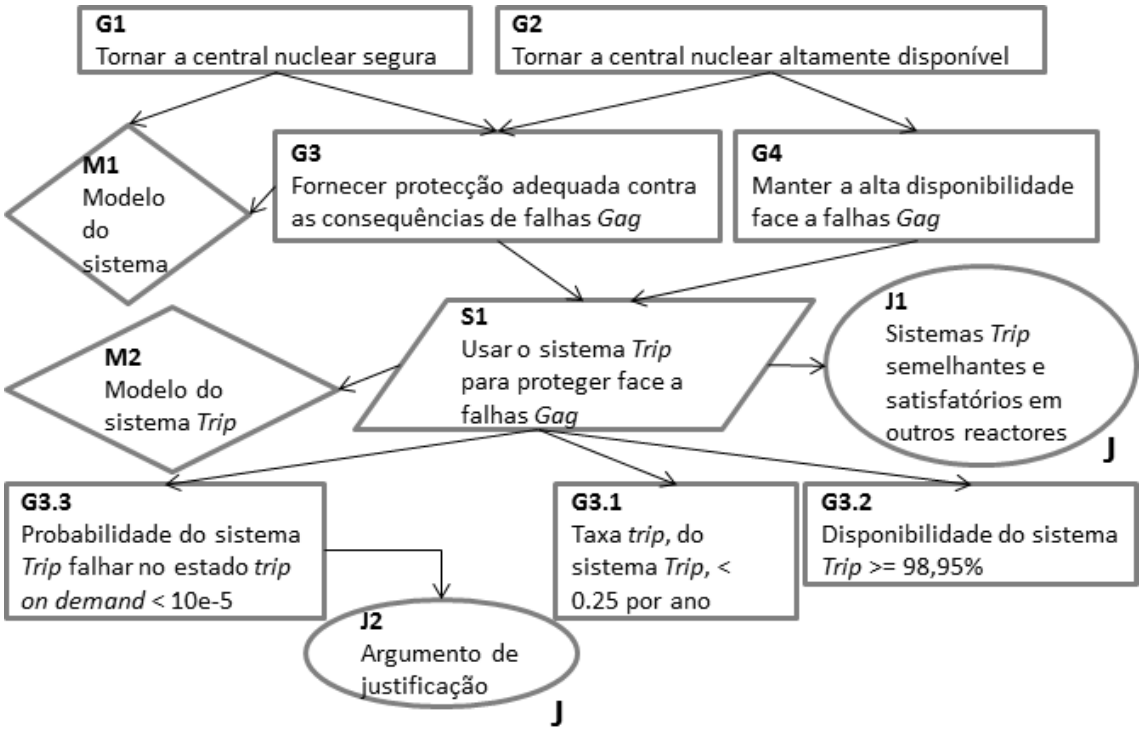


Figura 2.10: Exemplo da aplicabilidade da notação GSN [38]

Apesar do interesse que a abordagem GSN aparenta ter face a atributos de qualidade relacionados com a segurança, esta abordagem não perfaz o nosso interesse face a uma abordagem de validação de sistemas, que é, ser capaz de oferecer um mecanismo que ajude na definição de métricas para medir a complexidade e completude de modelos orientados a objectivos.

2.5 Sumário

Neste capítulo introduziram-se os conceitos de Engenharia de *Software*, Engenharia de Requisitos, e Engenharia de Requisitos Orientada a Objectivos. Foram apresentadas três das mais importantes metodologias de Engenharia de Requisitos Orientada a Objectivos: KAOS, *i* framework*, e NFR *framework*. Estas modelam, de formas diferentes, os requisitos dos sistemas de *software*.

Relativamente ainda a esta área da engenharia orientada a objectivos, viram-se também duas abordagens de validação de sistemas, *Goal-Question-Metric* e *Goal Structuring Notation*. A primeira é usada para definir métricas que

permitam atingir determinados objectivos de avaliação. E a segunda serve para modelar aspectos de segurança dentro dos sistemas de *software*.

3

Trabalho relacionado

Neste capítulo serão apresentadas três abordagens: AIRDoc, Métricas *i** para a integração da EROO com processos MDD, e *i*MDF$_M$. Depois será feita uma análise sobre estas abordagens. Há que considerar que cada uma das abordagens é ilustrada por exemplos recolhidos das suas descrições originais, e os quais apresentamos nesta secção a título de ilustração. Qualquer discussão mais aprofundada foge ao âmbito desta dissertação.

3.1 AIRDoc

AIRDoc, o acrónimo para ***A**pproach to **I**mprove **R**equirements **Doc**uments* (em português Aproximação para Melhorar Documentos de Requisitos), é um modelo que visa facilitar o processo de identificação de potenciais problemas sintáticos em documentos de requisitos utilizando refabricação[4] (do inglês *refactoring*) e padrões de requisitos. A abordagem, descrita por Ramos em [39, 40], utiliza o modelo GQM para avaliar modelos de requisitos (nomeadamente *use cases*). A abordagem AIRDoc consiste numa fase de avaliação, constituída pelos quatro primeiros passos, e numa fase de melhoria, onde estão envolvidos os dois últimos passos, tal como apresenta a Figura 3.1.

[4] Processo de modificar um sistema de *software* para melhorar a sua estrutura interna sem alterar o seu comportamento externo.

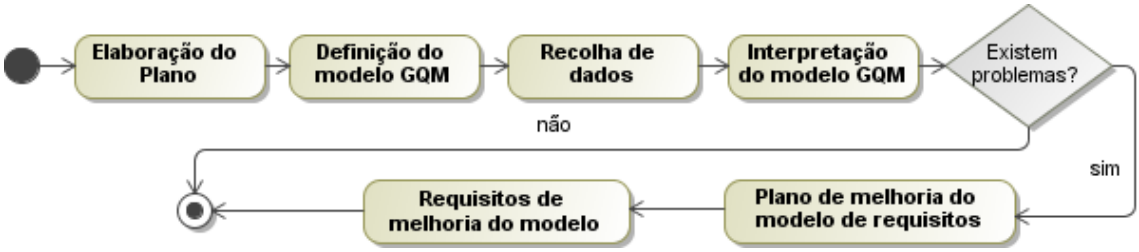


Figura 3.1: Diagrama de actividades do AIRDoc [39]

Para melhor entender este modelo, alguns dos *templates* definidos pelo AIRDoc serão apresentados com um exemplo. O caso de estudo definido em [39] trata de um sistema de análise de impostos da empresa SERPRO do Ministério das Finanças do Brasil. O artefacto escolhido é o modelo de requisitos *Adjustment Taxes*, que descreve a correcção do valor dos impostos pagos pelos cidadãos.

3.1.1 Passo1 – Elaboração do plano

A primeira fase da aproximação AIRDoc é a avaliação do modelo *use cases*, e de forma a garantir o sucesso dessa avaliação são executadas as seguintes tarefas: **(i) definição da equipa de qualidade** – a cada membro da equipa de qualidade seleccionado deve ser associado pelo menos um papel, para isso devem ser definidos os papéis necessários (por exemplo, engenheiro de requisitos, revisor, etc.), as responsabilidades associadas, e o responsável de cada um desses papéis; **(ii) escolha de ferramentas e outros recursos** – aqui o responsável pela escolha das ferramentas deve indicar, para cada ferramenta ou recurso, o propósito da sua utilização e o tempo de aprendizagem requerido; **(iii) estabelecer os requisitos de qualidade** – esta tarefa deve tornar claro os requisitos de qualidade que vão ser analisados, indicando, para cada um deles, a sua definição, o modelo de requisitos associado e os respectivos requisitos em análise (Tabela 3.1); **(iv) apresentar o plano de trabalho** – o plano de trabalho consiste num documento com as tarefas anteriores expostas de forma clara.

Tabela 3.1: *Template* para a definição dos requisitos de qualidade [39]

Objectivo	Modelo de requisitos	Requisito em análise	Atributo de qualidade
Melhorar a Sustentabilidade	*Adjustment Taxes*	Requisito *display* – referente aos *use cases*: (i) *Display spreadsheet control*; (ii) *Display screen of user analysis*.	Sustentabilidade – queremos saber o quão fácil é de manter e/ou adicionar novos recursos ao requisito *display*.

3.1.2 Passo 2 – Definição do modelo GQM

A segunda actividade da fase de avaliação prevê a definição de todas as medições que vão ser necessárias aplicar aos modelos de requisitos por forma a garantir os objectivos delimiados no passo anterior. Esta actividade é composta pelas seguintes tarefas: **(i) definição dos requisitos de medição** – consiste em detalhar a tarefa de especificação de requisitos de qualidade do Passo 1 (correspondente à Tabela 3.1); **(ii) elaboração de questões** – a Tabela 3.2 exemplifica o processo de elaboração de questões, estando estas associadas à operação dos requisitos; **(iii) definição de métricas** – na Tabela 3.3 são definidas as métricas associadas a cada questão; **(iv) elaboração de hipóteses** – a Tabela 3.4 apresenta as respostas possíveis às questões expostas. Olhando para estas tarefas à luz da abordagem GQM, elas correspondem, respectivamente, às fases de definição de objectivos (Nível Conceptual), elaboração de perguntas (Nível Operacional), e especficação de métricas (Nível Quantitativo), desta mesma abordagem (ver Secção 2.4.1). A discussão dos detalhes deste exemplo foge ao âmbito desta dissertação.

Tabela 3.2: *Template* para a definição das questões (parcial) [39]

Objectivo – Avaliar no *Adjustment Taxes* (modelo de requisitos) o requisito *display* para prever a qualidade de Sustentabilidade.			
Q1			Quão fácil é de entender o requisito *display*? **(Compreensão)**
	Q1.1		Como está o documento composto? **(Tamanho)**
		Q1.1.1	Quantos passos são necessários para especificar o requisito *display*?
		Q1.1.2	Quantos passos existem no documento de requisitos?
		Q1.1.3	Quantos *use cases* são necessários para especificar o requisito *display*?

Tabela 3.3: *Template* para a definição de métricas [39]

Métricas		Dados necessários
Q1.1.1	M1 – Quantos passos são necessários para especificar o requisito *display*?	Contar o número total de passos que descrevem o requisito *display*.
Q1.1.2	M2 – Quantos passos existem no documento de requisitos?	Contar o número total de passos que existem no documento de requisitos.
Q1.1.3	M3 – Quantos *use cases* são necessários para especificar o requisito *display*?	Contar o número de *use cases* onde existe pelo menos um passo que contribui para a especificação do requisito *display*.

Tabela 3.4: *Template* para a definição de funções e hipóteses que suportam a questão Q1 [39]

Q1	Quão fácil é de entender o requisito *display*? **(Compreensão)**
H1a	O requisito *display* é fácil de entender, porque o valor da Função H1 é menor que 0,04.
H1b	O requisito *display* é difícil de entender e/ou estender, porque o valor da Função H1 é maior que 0,04.
Função H1	$\frac{(M_1/M_3)}{M_2} \rightarrow$ esta função mostra a relação entre o tamanho médio dos passos dos *use cases* utilizados para descrever o requisito *display* e o tamanho de todos os *use cases* do modelo de requisitos. Assim, espera-se examinar como o tamanho dos *uses cases* é homogéneo.
Nota	Se o valor da função H1 estiver entre 0,01 e 0,04, então o tamanho do *use cases* utilizado para descrever o requisito *display* é aceitável.

3.1.3 Passo 3 – Recolha de dados

O objectivo da terceira actividade do processo de avaliação, para além da recolha dos dados, é o de garantir que as medições definidas pelas métricas especificadas encontram-se correctas. Para tal, esta actividade é composta por um **período de ensaios** e por um processo de **medições e recolha de dados**. No período de ensaios, as métricas definidas no Passo 2 são testadas, e os resultados obtidos são analisados para comparar com os limites impostos pela equipa de qualidade na Tabela 3.4 (estes limites podem ser modificados caso se veja ser necessário). Após este processo, os dados são recolhidos (Tabela 3.5).

Tabela 3.5: *Template* para a apresentação dos valores das métricas M1, M2 e M3 [39]

Métricas	Valores
M1 – Quantos passos são necessários para especificar o requisito *display*?	798
M2 – Quantos passos existem no documento de requisitos?	1533
M3 – Quantos *use cases* são necessários para especificar o requisito *display*?	2

3.1.4 Passo 4 – Interpretação do modelo GQM

A última actividade da fase de avaliação tem como objectivo a utilização dos dados, recolhidos no passo anterior, para responder às questões propostas e ao mesmo tempo verificar se os objectivos foram alcançados. Durante este passo, o que foi estimado nos passos anteriores será discutido pela equipa de qualidade de *software* numa **sessão de *feedback***. O propósito desta sessão é analisar e interpretar os dados obtidos com base nas hipóteses traçadas. Depois,

na tarefa de **resultados das medições**, deve-se escrever um relatório com todas as observações relevantes, conclusões e pontos de acção formulados durante a sessão de *feedback*. Em situações onde os dados possam apresentar possíveis problemas para os modelos de requisitos, deve ser criada uma tabela que indique cada problema identificado (Tabela 3.6). Caso nenhum problema seja encontrado a avaliação termina. Recordamos que não é do âmbito desta dissertação a discussão dos exemplos.

Tabela 3.6: *Template* de análise das hipóteses H1a e H1b [39]

Função H1	$\frac{(M_1/M_3)}{M_2} \rightarrow \frac{(798/2)}{1533} = \mathbf{0,26}$
Conclusão	A hipótese H1a é contestada, pois o valor da função H1 é maior que 0,04. Portanto, a hipótese H1b é suportada.
Respondendo à questão: "**Q1.** Quão fácil é de entender o requisito *display*?"	
"O requisito *display* é difícil de entender" (Hipótese **H1b**)	

3.1.5 Passo 5 – Plano de melhoria do modelo de requisitos

A fim de melhorar o modelo de requisitos é necessário descobrir quais os problemas do modelo. Para isso, o AIRDoc oferece um catálogo que relaciona os problemas/sintomas com as técnicas de refabricação ou de padrões apropriadas. Este catálogo oferece uma descrição do problema bem como as possíveis soluções. Após analisar cada problema individualmente, e a sua possível resolução, escolhe-se o padrão ou refabricação que mais se adequada à resolução pretendida.

3.1.6 Passo 6 – Requisitos de melhoria do modelo

Após a escolha dos padrões ou refabricações necessárias para resolver os problemas, devemos aplicar os processos descritos para cada situação ao modelo de requisitos. Esta última actividade começa por **aplicar as soluções selecionadas**, depois deve **armazenar os resultados** obtidos pelas soluções aplicadas, e por fim **comparar os modelos**. Para esta última é necessário voltar a realizar o Passo 4, e comparar os resultados obtidos antes e depois da utilização do AIRDoc.

3.2 Métricas *i** para a integração da EROO com processos MDD

Giachetti, *et al.*, em [41], apresentam um processo de integração de modelos MDD[5], o acrónimo para ***M**odel-**D**riven **D**evelopment* (em português Desenvolvimento Orientado a Modelos), com modelos *i**. Este processo envolve um conjunto de métricas que ajudam a validar os modelos *i** face a uma aproximação MDD particular, baseada nos modelos de classe UML. Os autores dividiram o processo de integração em cinco fases (Figura 3.2): formulação de métricas; declaração do metamodelo *i**; definição do modelo de validação *i**; especificação de métricas *i**; e geração de extensões *i**.

Figura 3.2. Processo de integração de métricas *i [41]**

De seguida iremos detalhar os diferentes passos para melhor compreender como os autores defendem que a transformação de modelos *i** para modelos MDD deve ser feita.

3.2.1 Passo 1 – Formulação de métricas

O primeiro passo consiste na formulação de métricas de validação apropriadas, e está dividido em duas etapas: identificação dos elementos da ferramenta *i** que participam na geração do modelo MDD; e definição das métricas. A Tabela 3.7 apresenta a relação entre os elementos *i** e os elementos do modelo de classes.

Tabela 3.7: Guia para a transformação de modelos *i para modelos de classes [41]**

Elementos *i**	Informação adicional	Elementos do Modelo de Classe
Actor		Classe
Recurso	Entidade física	Classe
	Entidade informativa relacionada a um recurso físico ou actor	Um atributo que representa a informação da classe gerada a partir do actor ou do recurso físico

[5] Abordagem de desenvolvimento de *software* onde os principais artefactos são modelos. Estes modelos são descrições de sistemas a partir de uma determinada perspectiva, omitindo detalhes irrelevantes para que as características de interesse sejam vistas de forma mais clara.

	Recurso em uma árvore de decomposição	Argumento de entrada para o serviço gerado a partir da tarefa relacionada
	Recurso *dependum*[6]	Argumento de entrada da tarefa *depender*
	Entidade física dentro dos limites de um actor	Uma associação entre as classes geradas a partir do recurso físico e do actor proprietário
Tarefa	Participa numa dependência de recursos como *depender* ou *dependee*	Um serviço da classe gerada a partir do recurso *dependum*
	Se gerar um recurso	Um serviço de criação da classe gerada a partir do recurso
Relação de Dependência	Onde o recurso *dependum* e os actores *depender* e *dependee* são transformados em classes	Associações são automaticamente definidas entre as classes geradas

A Tabela 3.8 apresenta a definição das métricas utilizando a aproximação GQM. Estas métricas podem ser divididas em dois grupos: métricas para avaliar a geração de classes e atributos (NTR, WAG, e ECG); e métricas para avaliar a geração de serviços (NIC, WSE, NIDR, e SWA).

Tabela 3.8: Aplicação do GQM para definir as métricas [41]

Objectivo	Melhorar a transformação dos elementos *i** em um processo de geração do modelo de classes a partir da perspectiva do analista de requisitos MDD.
Pergunta	**Métrica**
P1. Que recursos produzem uma geração errada do modelo de classes?	*Non-Transformable Resources* (NTR)
	Wrong Attributes Generation (WAG)
P2. Que recursos podem ser melhorados para a geração do modelo de classes?	*Empty Class Generation* (ECG)
	Non-Instantiable Classes (NIC)
	Non-Instantiable Dependum Resources (NIDR)
P3. Que tarefas produzem uma geração erradas do modelo de classes?	*Wrong Services Specification* (WSE)
P4. Que tarefas podem ser melhoradas para a geração do modelo de classes?	*Non-Instantiable Classes* (NIC)
	Non-Instantiable Dependency Resources (NIDR)
	Services Without Arguments (SWA)

[6] Elemento em torno do qual se centra a relação de dependência. Esta relação é composta por dois actor, *depender* e *dependee*. O *depender* depende do *dependee* na relação de dependência.

As métricas definidas na Tabela 3.8 podem ser:

- ***Non-Transformable Resources*** – identifica recursos que não são especificados como entidades físicas ou entidades informativas, já que a transformação de recursos *i** varia de acordo com o tipo de recurso envolvido;
- ***Wrong Attributes Generation*** – determina o número de recursos informativos que não estão relacionados com um recurso físico ou com um actor, para verificar a correcta geração dos atributos da classe;
- ***Empty Class Generation*** – demarca os recursos físicos e actores que não estão relacionados com recursos informativos, a fim de prever a geração de classes sem atributos;
- ***Non-Instantiable Classes*** – verifica o número de recursos físicos que não têm uma tarefa relacionada à sua produção, para verificar se o serviço que produz novas instâncias foi devidamente definido;
- ***Non-Instantiable Dependum Resources*** – esta métrica considera a identificação dos recursos físicos *dependum* que não têm uma tarefa de produção relacionada, a fim de verificar se as tarefas *dependee* relacionadas estão correctamente marcadas para a geração de *software*;
- ***Wrong Services Specification*** – identifica as tarefas *dependee* que produzem recursos diferentes dos recursos *dependum* relacionados, para verificar se essas tarefas irão gerar os serviços correctos;
- ***Services Without Arguments*** – calcula o número de tarefas terminais numa árvore de decomposição, que não tenha nenhum recurso relacionado, com o intuito de identificar argumentos que não sejam os argumentos criados por omissão (todos os serviços têm pelo menos um argumento que é a referência para o objecto que executou o serviço).

3.2.2 Passo 2 – Declaração do metamodelo *i**

O segundo passo corresponde em declarar o metamodelo alvo. Para tal, os autores basearam o seu metamodelo [41] noutras propostas de metamodelos *i** em [42, 43].

3.2.3 Passo 3 – Definição do modelo de validação *i**

A definição do modelo de validação *i** deve incluir a informação necessária para garantir a correcta aplicação das métricas, em particular informação que não conste no metamodelo *i** definido no ponto anterior. A Figura 3.3 representa o modelo de validação e apresenta o mapeamento da informação entre o modelo de validação e o metamodelo *i**.

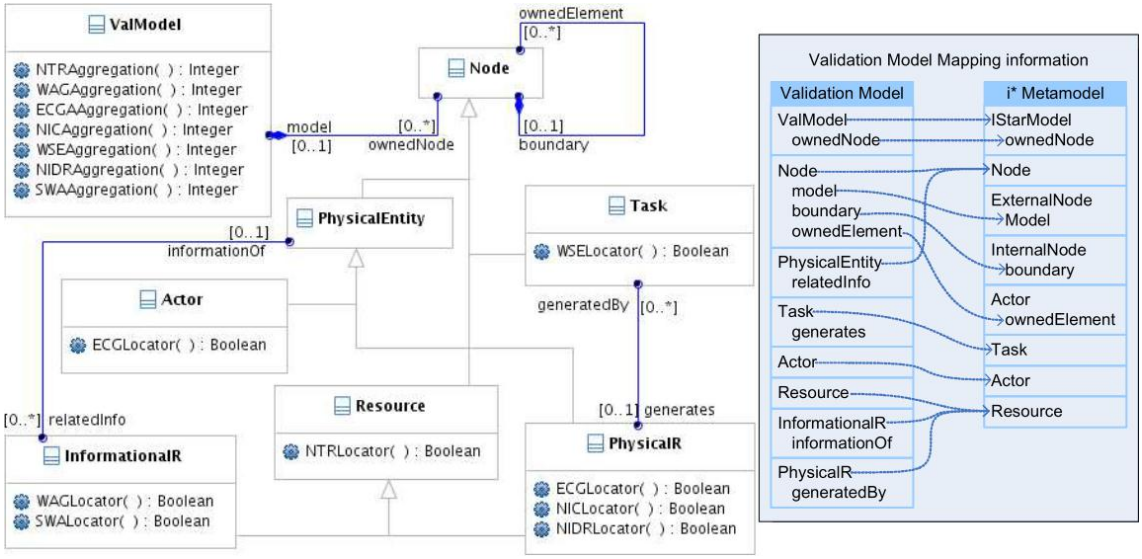


Figura 3.3: Modelo de validação [41]

3.2.4 Passo 4 – Especificação de métricas *i**

O quarto passo corresponde à especificação das métricas através da linguagem OCL (*Object Constraint Language*) e integração destas com o modelo de validação (Tabela 3.9). As classes da Figura 3.3 são compostas por atributos referentes às regras OCL, com os seus nomes e resultado de saída. Para a especificação de métricas são aplicados os padrões de métricas definidos em [44], nomeadamente: *aggregation*, *locator*, e *condition-checker*. Os padrões *locator* e *condition-checker* são utilizados para identificar os elementos envolvidos na avaliação das métricas, e o padrão *aggregation* é utilizado para retornar o valor numérico final.

A fim de completar a definição das métricas, é acrescentada uma nova propriedade que diz respeito ao nível de alerta relacionado, que pode ser: **crítico** – indica que a situação identificada pela métrica impede a transformação dos elementos envolvidos; **alerta** – identifica um problema de modelação que pode ser corrigido para melhorar a geração do modelo; **informação** – indica que

é possível adicionar informações adicionais ao modelo, de forma a melhorar o processo de geração de modelos.

Tabela 3.9: Exemplo da especificação de métricas com a linguagem OCL[7] [41]

Métrica	**Assunto de Medição**	**Nível de alerta**
M1: *Non-Transformable Resources* (NTR)	Recursos	Crítico
Context: Model::NTRAggregation() : Integer		
Body: *result = self.ownedNode->select(rs | rs.oclIsKindOf(Resource) and rs.NTRLocator())->size() + self.ownedNode.ownedElement->select(rs | rs.oclIsKindOf(Resource) and rs.NTRLocator())->size()*		
Context: Resource::NTRLocator() : Boolean		
Body: *result = (not self.oclIsKindOf(PhysicalResource) and not self->oclIsKindOf(InformationalResource))*		

3.2.5 Passo 5 – Geração de extensões *i**

Por último, o quinto passo utiliza o modelo de validação (Figura 3.3) e a especificação das métricas em OCL (Tabela 3.9) para gerar extensões, de forma a integrar as métricas propostas com a *framework i**. Estas extensões são geradas a partir da proposta apresentada em [45] para geração automática de perfil UML, que utiliza as informações de mapeamento da Figura 3.3.

3.3 *i*MDF$_M$

Franch e Grau, em [44], afirmam que "*ter um bom conjunto de métricas permite não só analisar a qualidade de um modelo individual, mas também comparar modelos alternativos relativamente a determinadas propriedades, de forma a escolher o modelo alternativo mais apropriado*". Com isto em mente, Franch e Grau definiram uma *framework* para a definição de métricas em modelos *i**, *i*MDF, o acrónimo para ***i**** ***M****etrics* ***D****efinition* ***F****ramework*. A ferramenta é composta por:

- **metamodelo para a *framework i**** - foi definido um metamodelo para a ferramenta *i** em que se representam os conceitos que iriam ajudar no processo de definição de métricas;
- ***template* de padrões** – ao analisar as métricas baseadas em *i** observaram-se algumas situações semelhantes e, como tal, definiu-

[7] As partes da especificação OCL sublinhadas e escritas a itálico devem ser substituídas de acordo com o estereótipo definido.

se um *template* para a definição de padrões para ajudar na especificação de métricas *i**;

- **catálogos de padrões** – as métricas podem ser caracterizadas de várias formas (por exemplo, métricas de declaração, métricas de definição, métricas de transformação, métricas de elementos auxiliares), nesta fase os autores catalogaram os padrões face às características das métricas, de forma a facilitar o uso dos padrões aquando da definição das métricas.

Com o intuito de melhor conduzir o analista na definição de métricas para modelos *i**, Franch criou o método *i*MDF$_M$ (***i**** ***M**etrics **D**efinition **F**ramework **M**ethod*) [46]. A Figura 3.4 apresenta os passos que compõem este método.

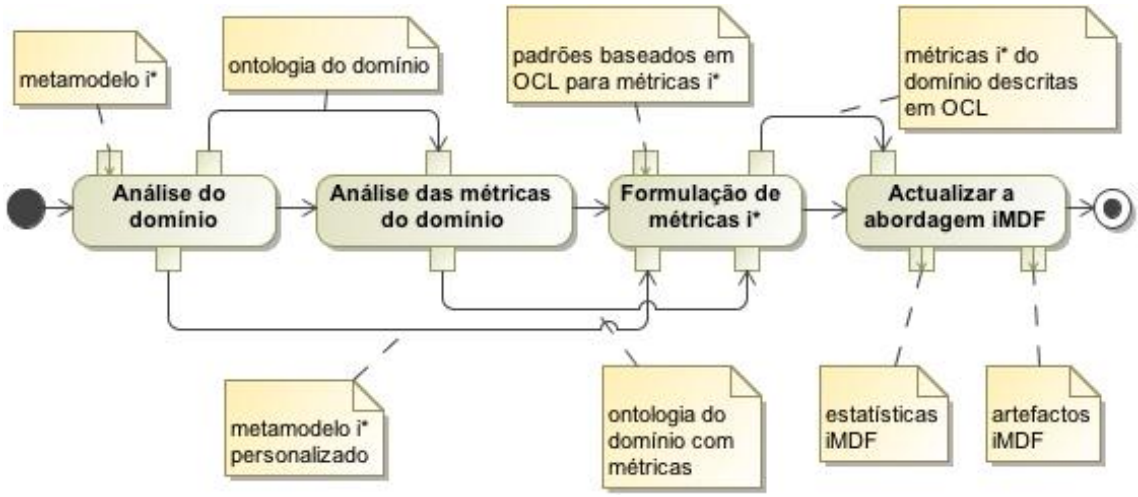


Figura 3.4: Diagrama de actividades do método iMDF$_M$ [47]

3.3.1 Passo 1 – Análise do domínio

Neste passo pretende-se compreender o domínio de forma a ser possível mapear os seus conceitos para o metamodelo da metodologia *i**. É composto por duas actividades: **criar uma ontologia do domínio** – a partir do conhecimento sobre o domínio é possível criar uma ontologia; **mapear a ontologia do domínio para o metamodelo *i**** - é estabelecida a relação entre os conceitos de domínio (por exemplo, processos de negócio, *stakeholders*, componentes do *software*, etc.) e os elementos *i** (Figura 3.5); **personalizar o metamodelo *i**** - o metamodelo definido é refinado numa especialização do domínio, sendo possível, adicionar novos atributos ou classes, e também impor novas restrições.

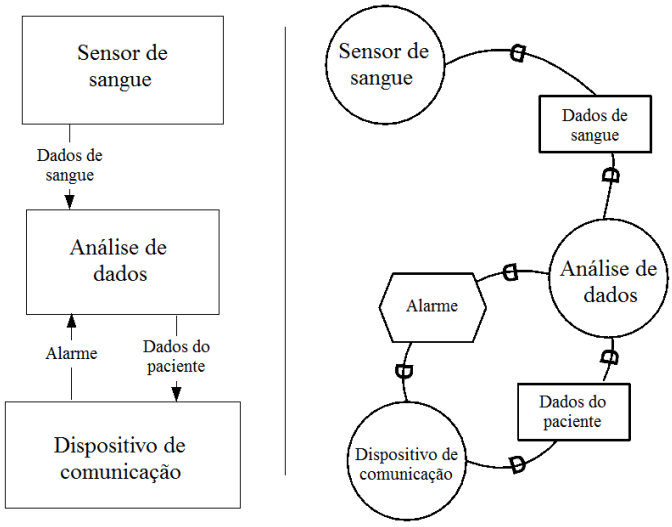


Figura 3.5: Exemplo do mapeamento de um diagrama arquitectural em blocos [47]

3.3.2 Passo 2 – Análise das métricas do domínio

Este passo tem como objectivo analisar o conjunto de métricas do domínio iniciais antes de serem formalizadas. As actividades realizadas são as seguintes: **estender a ontologia do domínio** – a ontologia do domínio é estendida de forma a incorporar os conceitos que não apareceram na análise anterior e que se tornam necessários para o uso do conjunto de métricas; **consolidar o domínio das métricas** – o conjunto de métricas é analisado em procura de inconsistências, falta de uniformidade, ambiguidades, entre outros problemas de estruturação, com o intuito de atribuir a cada métrica uma definição mais clara.

3.3.3 Passo 3 – Formulação de métricas *i**

Este passo torna as métricas operacionais em termos de construções *i**, tendo em conta o mapeamento entre a ontologia do domínio para o metamodelo *i**. É composto pelas seguintes actividades: **mapear as métricas sobre o metamodelo *i**** – a formulação das métricas é analisada e a sua definição alterada tendo em conta o metamodelo *i**; **expressar as métricas em OCL** – as métricas são tornadas operacionais através da reformulação da sua definição através do uso da linguagem OCL, tendo em conta o diagrama de classes e as restrições de integridade do metamodelo *i** (sempre que possível deve-se utilizar os padrões definidos no catálogo da abordagem *i*MDF).

3.3.4 Passo 4 – Actualizar a ferramenta *i*MDF

Por último, o resultado do processo é analisado a fim de melhor entender o método e a ferramenta *i*MDF por completo. Este passo é a combinação das seguintes actividades: **actualizar as estatísticas** – as estatísticas referem-se especialmente à aplicabilidade dos padrões que formam a língua; **actualizar a linguagem dos padrões** – após o acumular de algumas estatísticas, os padrões que raramente são utilizados podem vir a ser removidos ou reformulados; **actualizar o catálogo de métricas** – deve-se decidir se as métricas são adicionadas ou não ao catálogo de métricas, ou se é necessário remover alguma métrica desse mesmo catálogo.

3.4 Análise das abordagens

As abordagens apresentadas nas Secções 3.1, 3.2 e 3.3, contribuem para o âmbito desta dissertação, essencialmente ao nível da definição e especificação de métricas. Através deste estudo foi possível observar o trabalho que está a ser feito face à melhoria dos modelos *use cases*, ao intuito de melhorar os produtos de *software* através de integração de modelos *i** com modelos de classes, e à análise e comparação de modelos através de um catálogo de padrões sobre métricas *i**. A Tabela 3.10 mostra uma análise e resumo das abordagens estudadas.

Tabela 3.10: Análise das abordagens estudadas

	AIRDoc	**Métricas *i** para a integração da EROO com processos MDD**	***i*MDF$_M$**
Modelos suportados	*Use cases*	Modelos *i** e de classes	Modelos *i**
Formulação de métricas	GQM	GQM	Padrões *i*MDF
Definição de métricas	Linguagem natural, e hipóteses	OCL	OCL
Propósito	Avaliar e melhorar modelos de requisitos quanto à sua sustentabilidade e reusabilidade, identificando potenciais problemas	Integração de modelos *i** com processos MDD, com o intuito de melhorar a modularidade dos modelos *i**	Completar um catálogo de padrões sobre métricas *i** que permita analisar e comparar a qualidade de modelos

Resumo do processo	(i) indicação do modelo *use cases* em análise; (ii) definição de métricas; (iii) recolha de dados sobre a aplicação das métricas; (iv) escolha do mecanismo de refabricação ou de padrões; (v) aplicação do mecanismo	(i) guia de transformação dos modelos, (ii) identificação de métricas; (iii) declaração do metamodelo *i**; (iv) definição do modelo de validação; (v) especificação de métricas com OCL; (vi) geração de extensões entre modelos	(i) criar uma ontologia do domínio e mapeá-la sobre o metamodelo *i**; (ii) analisar as métricas do domínio e adapta-las à ontologia; (iii) formular as métricas com base no catálogo de padrões; (iv) actualizar o catálogo de padrões
Ponto fraco	(i) valores das hipóteses muito pouco claros, aparentam ser arbitrários; (ii) definição informal das métricas	Complexidade na aplicação das métricas propostas (sem suporte de uma ferramenta)	A definição de métricas fica sujeita ao catálogo existente, não havendo outra forma de as definir sem adicionar novos padrões ao catálogo
Ponto forte	(i) simplicidade do processo de avaliação e melhoramento; (ii) utilização de técnicas de refabricação e padrões; (iii) permite a avaliação de outros atributos de qualidade	(i) especificação de métricas em OCL, permitindo uma automatização directa; (ii) geração de modelos de classe UML a partir de modelos *i**	(i) especificação de métricas em OCL, permitindo uma automatização directa; (ii) uma vez que o catálogo de padrões foi validado, as métricas deixam de estar sujeitas a erros

A grande diferença entre estas abordagens, e de real interesse para o objectivo desta dissertação, é a forma que cada uma das abordagens tem para criar padrões de métricas, uma através de catálogos especificados por linguagem natural, as outras através de OCL. Apesar da linguagem natural ser mais simples e fácil de compreender, o facto de termos uma linguagem formal (por exemplo, OCL) é um ponto muito positivo pois possibilita a automatização do processo de identificação das métricas. Posto isto, as ideias analisadas podem ser um forte contributo para o âmbito desta dissertação. Contudo, outras formas de definir padrões de métricas serão interessantes de explorar, tal como o uso de XML para esse efeito [48].

Dado que não foi encontrado nenhum trabalho efectivo relativo à melhoria da qualidade de modelos KAOS, avançamos com a nossa abordagem sobre este tipo de modelos. Para além disso, estes modelos encontram-se bem representados, tendo em conta o suporte da ferramenta industrial Objectiver. A

importância que Lamsweerde deu à abordagem KAOS ao lançar o seu último livro [17], é também um forte contributo para o desenvolvimento de *software* orientado a objectivos.

3.5 Sumário

Ao longo deste capítulo foram apresentadas ao pormenor três abordagens preocupadas em avaliar a qualidade de modelos de requisitos.

A primeira, AIRDoc, formulada por Ramos, é uma abordagem composta por seis passos: elaboração do plano, especificando o objectivo que se pretende tratar e relativamente a que modelo de requisitos ou a que partes desse modelo; definição de métricas com auxílio da abordagem GQM; recolha de dados; interpretação de resultados; plano de melhoria do modelo, consistindo em catálogos de refabricações e padrões conforme o problema listado; e a aplicação desse plano de melhoria.

A segunda é uma aproximação de integração de métricas *i** com processos MDD, de Giachetti, *et al.*, que defende que modularização da ferramenta *i** contribui para a melhoria da qualidade dos seus modelos. Esta aproximação baseia-se na abordagem GQM para definir as suas métricas e na linguagem OCL para definir padrões de métricas.

A terceira, apresentada por Franch, consiste na definição de um método que ajuda na definição de métricas para modelos *i**. Esta aproximação é uma continuação ao seu trabalho e de Grau sobre a *framework i*MDF, e baseia-se em OCL para a definição de padrões de métricas. Franch afirma que ao obter um catálogo de padrões sobre métricas *i** completo ajudaram no processo de avaliação de modelos *i**, bem como comparar com outros modelos.

Depois da apresentação das metodologias, estas foram analisadas face às contribuições que poderão trazer a esta dissertação.

4

Métricas para a avaliação de modelos de objectivos KAOS

Este capítulo encontra-se estruturado da seguinte forma: introdução ao conjunto de métricas; descrição do modularKAOS; e definição de métricas. Na Secção 4.1 introduzimos as métricas alvo de estudo, que nos irão ajudar a avaliar os modelos de objectivos KAOS. Na Secção 4.2, discutimos a importância do modularKAOS no âmbito desta dissertação, bem como algumas modificações essenciais para a sua correcta integração com as métricas. Por fim na Secção 4.3, as métricas apresentadas vão ser estruturadas e formalizadas de acordo com o metamodelo actualizado da ferramenta modularKAOS.

4.1 Introdução ao conjunto de métricas

O nosso objectivo é analisar modelos de objectivos KAOS com respeito à sua completude e complexidade. Portanto, seguindo a abordagem GQM, o nosso nível conceptual é composto pelos objectivos de analisar a complexidade e completude destes modelos.

O processo de construção das métricas de complexidade propostas nesta secção foi inspirado por métricas de complexidade de *software* propostas noutros contextos (por exemplo, Desenho Orientado a Objectos [49]).

A Tabela 4.1 e a Tabela 4.2 sintetizam o resultado da aplicação da abordagem GQM para propor um conjunto de métricas que permitam satisfazer os objectivos de avaliação da completude e da complexidade.

Na Tabela 4.1 temos o modelo GQM para alcançar a completude dos modelos de objectivos KAOS. Para isto foram formuladas cinco perguntas, que

dizem respeito à: (P1) atribuição da responsabilidade dos objectivos a agentes; (P2) associação de objectos a objectivos; (P3) provisão de resoluções de obstáculos; (P4) associação de operações a objectivos; e (P5) associação de operações a agentes.

Tabela 4.1: GQM para a avaliação da completude

Objectivo	Avaliar a completude dos modelos de objectivos KAOS do ponto de vista do engenheiro de *software*
Pergunta	**Métrica**
P1. Quão perto estamos de completar a atribuição de todas as responsabilidades de um objectivo a agentes?	**M1.** Percentagem de objectivos folha que têm um agente associado.
P2. Quão detalhado é o modelo de objectivos em respeito aos objectos?	**M2.** Percentagem de objectivos folha que têm um objecto associado.
P3. Quão perto estamos de completar o modelo face a todos os obstáculos encontrados?	**M3.** Percentagem de obstáculos folha que têm uma solução associada.
P4. Quão detalhado é o modelo de objectivos em respeito às operações?	**M4.** Percentagem de objectivos folha que têm uma operação associada.
P5. Quão bem suportadas estão as operações do modelo de objectivos?	**M5.** Percentagem de operações que têm um agente associado.

Relativamente à complexidade dos modelos alvo, foram definidas quatro questões, mostradas na Tabela 4.2, que visam medir: (P6) a quantidade de responsabilidade suportada pelos agentes do modelo; (P7) o número de objectos associados aos objectivos folha; (P8) a dificuldade de compreensão dos modelos de objectivos com respeito aos diferentes níveis de refinamento, ou seja, altura da hierarquia de objectivos; e (P9) o número de refinamentos de um objectivo.

Tabela 4.2: GQM para a avaliação da complexidade

Objectivo	Avaliar a complexidade dos modelos de objectivos KAOS do ponto de vista do engenheiro de *software*
Pergunta	**Métrica**
P6. Será que um agente tem demasiada responsabilidade no modelo?	**M6.** Número de objectivos folha associados a um agente específico.
	M7. Número mínimo de objectivos folha associados a um agente.

	M8. Número máximo de objectivos folha associado a um agente.
	M9. Número médio de objectivos folha associados a um agente.
P7. Será que um objectivo folha tem um número adequado de objectos associados?	**M10.** Número de objectos associados a um objectivo folha específico.
	M11. Número mínimo de objectos associados a um objectivo folha.
	M12. Número máximo de objectos associados a um objectivo folha.
	M13. Número médio de objectos associados a um objectivo folha.
P8. Quão difícil é de entender um objectivo folha, com respeito aos seus objectivos pai?	**M14.** Profundidade da hierarquia de objectivos.
P9. Quão complexo é um objectivo, com respeito aos seus refinamentos?	**M15.** Número de sub-objectivos, directos ou indirectos, de um objectivo.

Com o intuito de ajudar na formalização destas métricas, bem como na recolha dos dados para avaliação dessas mesmas métricas, traçámos alguns pressupostos:

i. A atribuição de responsabilidades de objectivos a agentes é herdada pelos objectivos resultantes do refinamento do objectivo alvo.

ii. A resolução de um obstáculo pode ser obtida de duas formas: atribuição directa a um objectivo (objectivo resolução); herança de resoluções provenientes dos obstáculos que o geram. Se o obstáculo for refinado em vários obstáculos, este só tem resolução se todos os seus refinamentos tiverem uma resolução associada.

iii. Equivalente ao declarado em (i), os agentes responsáveis por um objectivo são responsáveis por todos os objectivos resultantes do refinamento do objectivo alvo.

iv. A altura dos modelos é calculada segundo os refinamentos de objectivos, a declaração de obstáculos, o refinamento desses mesmos obstáculos, e as suas resoluções.

v. Os refinamentos directos de um objectivo são aqueles em que esse objectivo é o pai dos objectivos resultantes. Os refinamentos indirectos de um objectivo são todos aqueles que podem resultar de um primeiro refinamento desse mesmo objectivo. E isso inclui as resoluções dos obstáculos.

Antes de continuarmos com a definição e formalização do conjunto de métricas apresentado, discutimos primeiro o modularKAOS por forma a melhor enquadrar o problema.

4.2 modularKAOS

Actualmente, a avaliação feita à complexidade e completude de modelos KAOS é, normalmente, apenas qualitativa. Nesta dissertação, propomos uma quantificação dessas avaliações, de modo a torná-las repetíveis de forma consistente e automatizada. Após a definição do modelo GQM, surgiu a necessidade de encontrar uma ferramenta que nos possibilitasse a recolha de toda a informação relevante dos modelos (por exemplo, quantidade de agentes e objectivos no modelo) para o cálculo das métricas. Tal como descrito na Secção 2.3.1, forram consideradas duas ferramentas candidatas: o Objectiver e o modularKAOS.

O modularKAOS [11, 12] tornou-se uma opção interessante pois os modelos estão acompanhados de um ficheiro XML que descreve, de forma textual, todos os elementos e conexões do modelo, e também pela possibilidade de integração do OCL no metamodelo da ferramenta, permitindo uma avaliação eficaz dos modelos. Estas propriedades advém dos plugins EMF (*Eclipse Modeling Framework*) [50] e GMF (*Graphical Modeling Framework*) [51], do IDE Eclipse, que implementam e desenham a ferramenta.

O Objectiver, revelou-se uma opção menos adequada ao nosso contexto dado que não permite uma leitura externa (por exemplo, usando um parser) do conteúdo dos seus modelos, visto que os ficheiros gerados encontram-se num formato desconhecido. Para além disso, por ser uma ferramenta comercial, tem os seus recursos limitados para versões gratuitas.

Sendo assim, a opção da ferramenta de base sobre a qual implementámos a nossa ferramenta de recolha automática das métricas recaiu sobre o modularKAOS. Mais aspectos sobre o modularKAOS podem ser encontrados no Anexo A, onde apresentamos o metamodelo da versão desenvolvida por Monteiro [12], e também uma descrição dos elementos que o compõem.

4.2.1 Introdução às técnologias usadas

Antes de continuarmos com detalhes sobre o modularKAOS será interessante oferecer algum enquadramento sobre as linguagens utilizadas. Como já referimos, a nossa ferramenta foi implementada e desenha pelos plugins EMF e GMF do IDE Eclipse. A primeira *framework* oferece um metamodelo (Ecore) para descrever modelos e permite que a estes modelos sejam adicionadas restrições através da linguagem OCL (OCLInEcore) [52]. O Ecore pode também ser definido através de uma linguagem textual, Emfatic [53].

O GMF utiliza os elementos do metamodelo (Ecore ou Emfatic), fornecido pelo EMF, e cria um editor gráfico. A Figura 4.1 apresenta o fluxo de desenvolvimento do editor gráfico (a imagem é gerada pelo GMF *Dashboard* – prespectiva dos projectos EMF/GMF).

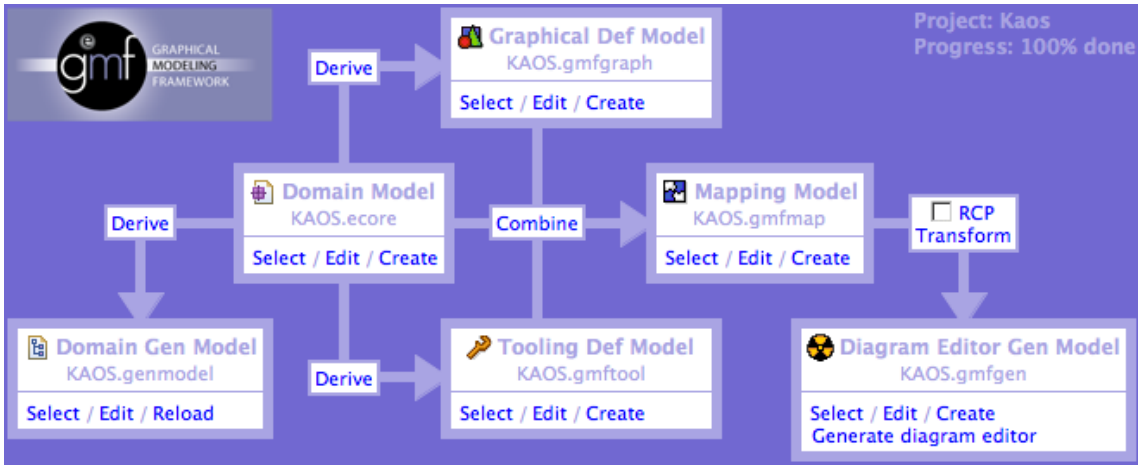


Figura 4.1: Fluxo de desenvolvimento do editor gráfico

Os ficheiros *gmfgraph*, *gmftool*, e *gmfmap*, são gerados automaticamente apartir dos dados do ficheiro *ecore*, e podem ser editados manualmente ou através de um linguagem própria, designada EOL (*Epsilon Object Language*) [54]. Esta é uma linguagem de programação imperativa para a criação e modificação de modelos EMF. Este tipo de ficheiros também são ficheiros de entrada para a criação do editor gráfico.

O editor gráfico GMF permite a validação dos elementos nele desenhados. Essa validação é feita ao nível do Ecore, mas também através da linguagem EVL (*Epsilon Validation Language*) [55], que trabalha sobre a linguagem EOL e permite a especificação de restrições bastante semelhantes à linguagem OCL. Desde o momento que a validação do editor gráfico é activada, este reconhece automaticamente ficheiros *evl*.

4.2.2 Modificações sobre a ferramenta modularKAOS

Ao longo do nosso trabalho sobre o modularKAOS foram surgindo situações de cariz funcional e visual que mereceram a nossa atenção para uma possível alteração da ferramenta. Preocupados com a correcta recolha e avaliação das métricas definidas para os modelos KAOS, tornou-se imprescindível a inclusão de uma raiz única à classe *KAOS* (Figura 4.2). Como os modelos KAOS pretendem resolver um único problema, sendo que a sua modelação começa por definir esse problema da forma mais abstracta possível, obrigar que a nossa ferramenta tenha apenas um objectivo inicial vai a favor das especificações do KAOS. Com isto evitamos problemas na recolha de métricas, visto que estas são referentes a um único objectivo inicial e não a um conjunto de objectivos.

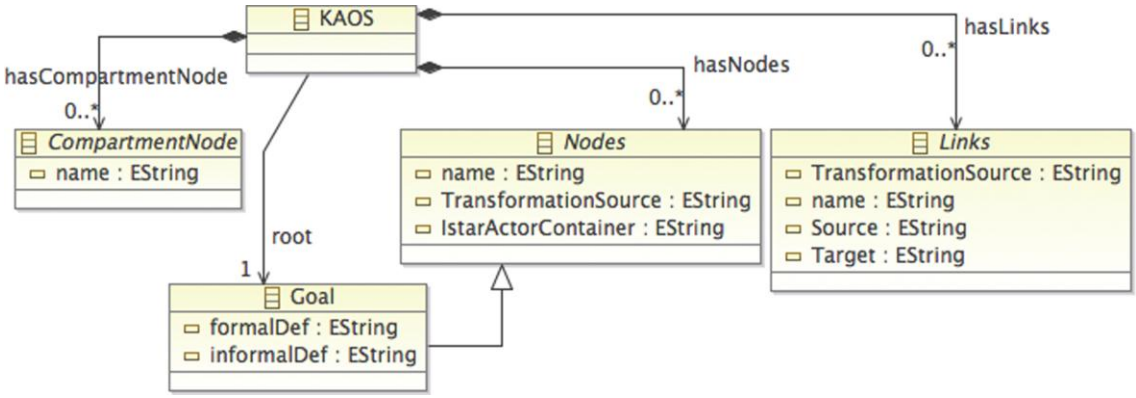


Figura 4.2: Alterações na classe *KAOS*

Para além disto, foi também necessário acrescentar um novo tipo de ligações que permitisse exprimir a relação entre um agente e uma operação. Foi então criada, uma nova subclasse *PerformsLinks* que herda os atributos da classe abstracta *Links*. Esta nova classe cria uma ligação de responsabilidade sobre a execução de um agente face a uma operação (Figura 4.3), o que nos possibilita avaliar a completude dos modelos KAOS relativamente a este factor.

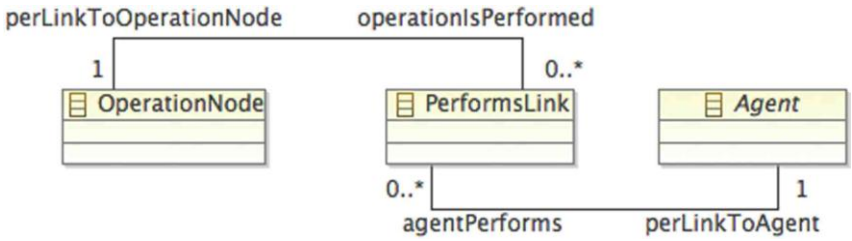


Figura 4.3: Nova classe *PerformsLink*

Tendo em conta que tanto os requisitos como as expectativas podem ainda ser refinados, deixa de fazer sentido obrigar estes objectivos a terem um agente

associado. Ao invés disso, é necessário que os objectivos folha, quer sejam requisitos ou expectativas, tenham pelo menos um agente associado. Como tal, modificámos as classes *Requirement* e *Expectation* para que os requisitos e expectativas pudessem ter zero ou mais agentes associados (Figura 4.4).

Utilizámos a linguagem EVL para verificar se os objectivos folha têm agentes, e caso não tenham, esta lança o respectivo aviso. Para isso, foi necessário especificar algumas restrições para esse fim.

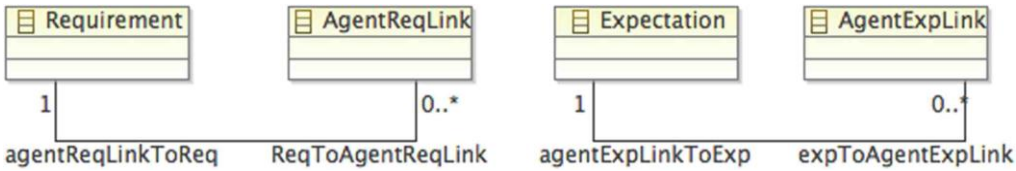


Figura 4.4: Alterações nas classes *Requirement* e *Expectations*

Tanto os objectivos como os obstáculos são elementos que servem para melhor representar o objectivo inicial. Os objectivos vão decompondo o objectivo inicial em objectivos mais específicos, enquanto que os obstáculos descrevem problemas nesses objectivos. Quer para avaliar a complexidade dos modelos relativamente à sua altura, quer para descobrir quão bem estão os obstáculos suportados por soluções, é necessário que a classe *Obstacle* esteja bem detalhada. Foi então necessário incluir dois atributos adicionais nesta classe (Figura 4.5):

- *obstIsRefinement*, referência para a classe *ObstacleRefinement*, que indica os refinamentos de obstáculos onde este obstáculo é um refinamento, ou seja, é o filho;
- *obstIsParent*, referência para a classe *ObstacleRefinement*, que indica os refinamentos de obstáculos onde este obstáculo é o pai.

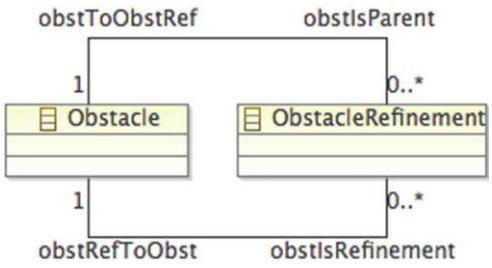


Figura 4.5: Alterações na classe *Obstacle*

No Anexo B é possível consultar a nova versão do metamodelo da ferramenta modularKAOS, que contém as alterações acima descritas.

Dada a falta de informação relativamente às dependências das versões dos plugins utilizados na construção da ferramenta modularKAOS, e sendo que os plugins EMF e GMF se encontram em constante actualização, foi-nos impossível integrar correctamente o projecto modularKAOS. Por esse motivo, implementámos de raiz esta ferramenta com base no que já tinha sido feito, e melhorámos alguns aspectos de forma a facilitar uma futura actualização do projecto, nomeadamente:

- especificação dos elementos gráficos no ficheiro Emfatic;
- integração de algumas restrições OCL no ficheiro Ecore;
- utilização da linguagem EOL para completar as formatações do ficheiro Emfatic.

Em relação aos dois primeiros pontos, é necessário referir que a partir do ficheiro Ecore é possível gerar o ficheiro Emfatic, e vice-versa. Portanto, qualquer modificação feita num destes ficheiros é facilmente transposta para o outro. Posto isto, voltamos a referir que o editor GMF é automaticamente gerado pelas especificações impostas num destes ficheiros, Emfatic ou Ecore. Assim sendo, para facilitar todo o processo de criação do editor, aquando de uma qualquer alteração no metamodelo ou na aparência dos elementos gráficos, é importante que essas modificações sejam feitas num destes ficheiros (Emfatic ou Ecore).

Desta forma, todos os elementos gráficos que, na versão da ferramenta de Monteiro em [12], eram declarados nos ficheiros *gmfgraph*, *gmftool* e *gmfmap*, que compõem o editor GMF, nós definimo-los no ficheiro Emfatic. A utilização da linguagem EOL vem colmatar os aspectos não suportados pelo ficheiro Emfatic, como por exemplo, definição de novos elementos decorativos, e manipulação de alguns aspectos dos ficheiros *gmfgraph*, *gmfmap* e *gmftool*.

No Anexo C é explicado como o projecto pode ser importado e quais os plugins e respectivas versões que permitem tirar o maior partido daquilo que já foi feito sobre esta ferramenta.

4.2.3 Interpretador modularKAOS

Segundo Lamsweerde, em [17], uma forma de garantir a qualidade dos requisitos é realizando perguntas a uma base de dados de requisitos que armazena as suas especificações. Para isto ser possível é necessário que os requisitos estejam representados por forma de diagramas, como por exemplo,

Diagrama de Classes. As perguntas à base de dados permitirão uma análise quanto à consistência e complexidade dos requisitos.

Apesar de nos encontrarmos em posição de realizar esse tipo de *queries* (devido às propriedades dos plugins EMF e OCLInEcore) e tendo-o feito para verificar a veracidade das formalizações em OCL das métricas, utilizar este processo para um conjunto de modelos é bastante desgastante e demorado (embora não tanto como uma análise visual aos modelos). Desta forma, tornou-se imprescindível termos um interpretador dos modelos gerados pelo modularKAOS. Sendo a geração de ficheiros XML com toda a informação dos modelos uma tão desejada característica, e um dos motivos de escolha da ferramenta modularKAOS, criámos em Java um *parser* desses ficheiros. Guardámos toda a informação em estruturas de dados e, uma vez terminada a leitura do ficheiro, retornámos toda a informação necessária, tal como o cálculo das métricas, e listagens dos dados.

Através da informação obtida com este interpretador dos ficheiros XML gerados pelo modularKAOS, foi-nos possível fazer a uma avaliação cuidada das métricas definidas, como discutiremos no Capítulo 5.

4.3 Definição de métricas

Baseando-nos na Tabela 4.1 e na Tabela 4.2, que ditam o GQM para a avaliação dos modelos de objectivos KAOS no que toca à sua completude e complexidade, e na Figura 4.6, onde é apresentado um excerto do metamodelo da ferramenta modularKAOS, foi-nos possível definir e formalizar o conjunto de métricas desejado.

Da Tabela 4.3 à Tabela 4.7 são apresentadas algumas métricas que respondem a um conjunto de questões (Tabela 4.1) relacionadas com a avaliação da completude dos modelos de objectivos KAOS, e da Tabela 4.8 à Tabela 4.11 definimos o conjunto de métricas que retratam a avaliação da complexidade destes modelos (relacionado com a Tabela 4.2). Para cada tabela é apresentada a respectiva questão, uma definição informal das métricas especificadas para responder à questão, e a sua definição formal utilizando OCL sobre o metamodelo apresentado na Figura 4.6. Sempre que necessário, incluiremos a definição formal das pré-condições para a computação das métricas. De forma a tornar as métricas mais claras, incluímos também nestas tabelas algumas métricas auxiliares directamente relacionadas com as métricas principais.

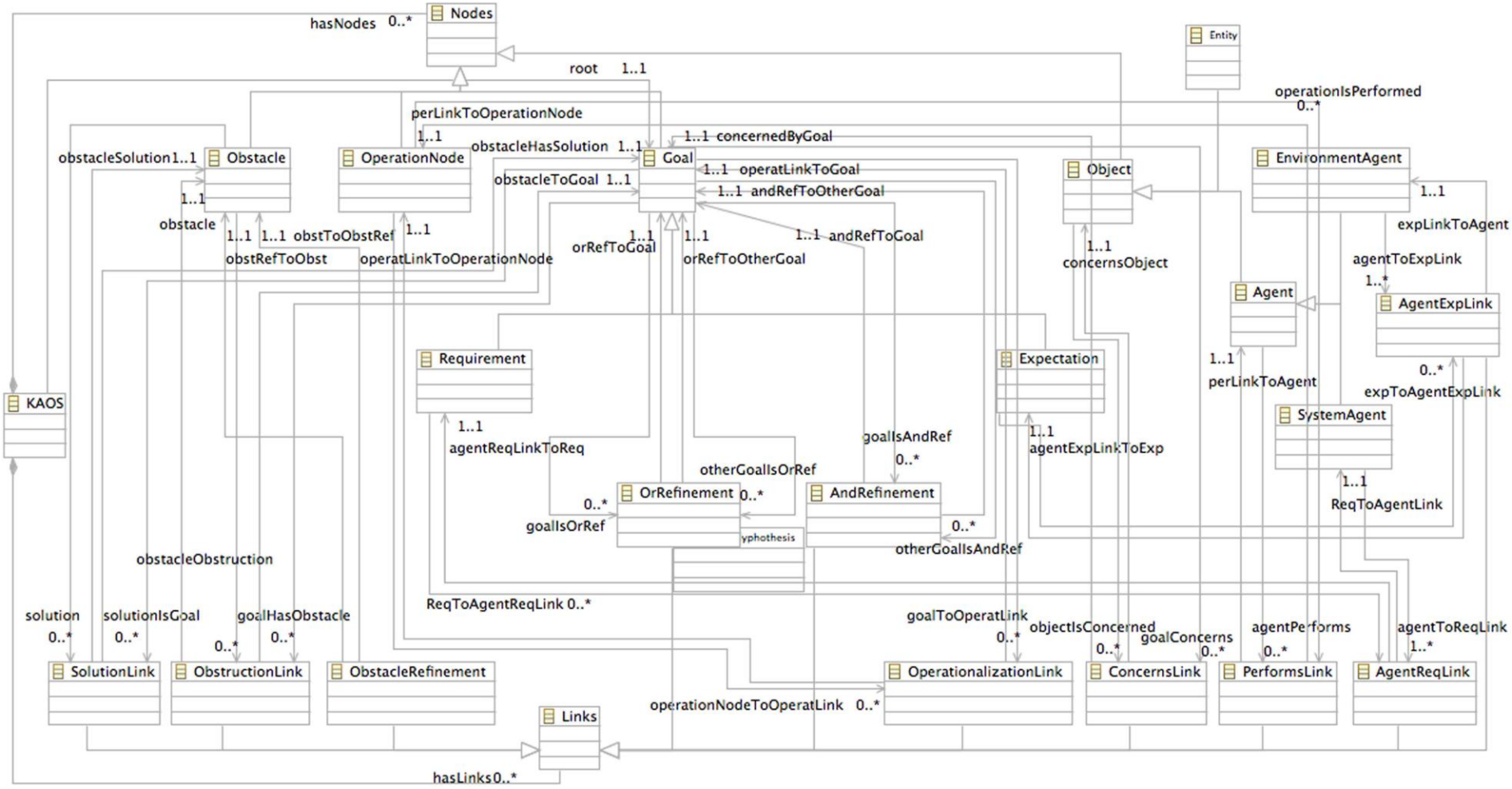


Figura 4.6: Excerto do metamodelo da ferramenta modularKAOS

4.3.1 Métricas para completude

Para garantir a completude dos modelos de objectivos, cada objectivo folha deve ser atribuído à responsabilidade de pelo menos um agente [17]. A métrica PLGWA na Tabela 4.3 aborda este problema. Para além disso, esta métrica também suporta o caso em que um objectivo pai é atribuído a um agente, e aí, todos os seus sub-objectivos também serão atribuídos a esse mesmo agente.

Tabela 4.3: Métricas que satisfazem o objectivo de completude da pergunta P1

P1 – Quão perto estamos de completar a atribuição de todas as responsabilidades de um objectivo a agentes?	
Nome	**PLGWA – *P**ercentage of **L**eaf **G**oals **W**ith an **A**gent*** (Percentagem dos Objectivos Folha com um Agente)
Definição informal	Percentagem de objectivos folha que têm, no modelo, pelo menos um agente associado.
Definição formal	***context KAOS*** ***def: PLGWA(): Real*** = *self.NLGWA() / self.NLG()*
Pré-condição	*context KAOS::PLGWA()* *pre: self.NLG() > 0*
Comentários	Se não existem objectivos folha o resultado é indefinido.
Métricas auxiliares	
Nome	**NLG – *N**umber of **L**eaf **G**oals*** (Número de Objectivos Folha)
Definição informal	Número de objectivos folha no modelo.
Definição formal	***context KAOS*** ***def: NLG(): Integer*** = *self.hasNodes->select(n: Nodes |* *n.oclIsKindOf(Goal) and n.oclAsType(Goal).ILG())->size()*
Nome	**NLGWA– *N**umber of **L**eaf **G**oals **W**ith an **A**gent*** (Número de Objectivos Folha com um Agente)
Definição informal	Número de objectivos folha que têm, no modelo, pelo menos um agente associado.
Definição formal	***context KAOS*** ***def: NLGWA(): Integer*** = *self.hasNodes->select(n: Nodes |* *n.oclIsKindOf(Goal) and n.oclAsType(Goal).ILG() and* *n.oclAsType(Goal).GNA() > 0)->size()*

Na Tabela 4.4 especificamos uma medida sobre o nível de detalhe baseada na identificação de objectos do sistema. Apesar de a associação de objectos com objectivos folha não ser uma regra obrigatória para os modelos de objectivos KAOS, esta fornece informação valiosa que pode ser utilizada nas seguintes fases de desenvolvimento do sistema. Assim sendo, trata-se de uma regra que traz uma forte contribuição para a completude dos modelos.

Tabela 4.4: Métricas que satisfazem o objectivo de completude da pergunta P2

P2 – Quão detalhado é o modelo de objectivos em respeito aos objectos?	
Nome	**PLGWO** – ***P**ercentage of **L**eaf **G**oals **W**ith an **O**bject* (Percentagem de Objectivos Folha com um Objecto)
Definição informal	Percentagem de objectivos folha que têm, no modelo, pelo menos um objecto associado.
Definição formal	***context KAOS*** ***def: PLGWO(): Real*** = *self.NLGWO() / self.NLG()*
Pré-condição	*context KAOS::PLGWO()* *pre: self.NLG() > 0*
Comentários	Se não existem objectivos folha o resultado é indefinido.
Métricas auxiliares	
Nome	**NLGWO** – ***N**umber of **L**eaf **G**oals **W**ith an **O**bject*
Definição informal	Número de objectivos folha que têm, no modelo, pelo menos um objecto associado.
Definição formal	**context KAOS** **def: NLGWO(): Integer** = self.hasNodes->select(n: Nodes | n.oclIsKindOf(Goal) and n.oclAsType(Goal).ILG() and n.oclAsType(Goal).GNO() > 0)->size()

Os obstáculos descrevem situações onde um objectivo pode não ser satisfeito, o que pode levar a falhas no sistema [27]. Lidar com estes obstáculos é muito importante porque reduz a probabilidade de falhas no sistema por causas conhecidas. Assim sendo, verificar que cada obstáculo tem uma resolução é uma métrica que descrevemos sobre a completude dos modelos de objectivos KAOS (Tabela 4.5).

Tabela 4.5: Métricas que satisfazem o objectivo de completude da pergunta P3

P3 – Quão perto estamos de completar o modelo face a todos os obstáculos encontrados?	
Nome	**PLOWS – *P**ercentage of **L**eaf **O**bstacles **W**ith a goal re**S**olution*** (Percentagem de Obstáculos Folha com uma Resolução)
Definição informal	Percentagem de obstáculos folha que têm, no modelo, pelo menos uma resolução associada.
Definição formal	***context KAOS*** ***def: PLOWS(): Real*** = *self.NLOWS() / self.NLO()*
Pré-condição	*context KAOS::PLOWS()* *pre: self.NLO() > 0*
Comentários	Se não existem obstáculos folha o resultado é indefinido.
Métricas auxiliares	
Nome	**NLO – *N**umber of **L**eaf **O**bstacles*** (Número de Obstáculos Folha)
Definição informal	Número de obstáculos folha no modelo.
Definição formal	***context KAOS*** ***def: NLO(): Integer*** = *self.hasNodes->select(n: Nodes |* *n.oclIsKindOf(Obstacle) and n.oclAsType(Obstacle).ILO())->size()*
Nome	**NLOWS – *N**umber of **L**eaf **O**bstacles **W**ith a re**S**olution*** (Número de Obstáculos Folha com uma Resolução)
Definição informal	Número de obstáculos folha que têm, no modelo, pelo menos uma resolução associada.
Definição formal	***context KAOS*** ***def: NLOWS(): Integer*** = *self.hasNodes->select(n: Nodes | n.oclIsKindOf(Obstacle)* *and n.oclAsType(Obstacle).ILO() and* *n.oclAsType(Obstacle).ONS()> 0)->size()*

Na Tabela 4.6 descrevemos outro nível de detalhe agora com respeito às operações. As operações representam soluções bem definidas que cumprem os objectivos. Se um objectivo é operacionalizado, então a forma como este vai ser realizado já está determinada. Caso contrário, é da responsabilidade da equipa da fase de desenvolvimento seguinte, desenhar uma solução para o objectivo, tendo total liberdade para o fazer. A métrica PLGWOp pode ser usada para avaliar até que ponto os objectivos do modelo são realizados.

Tabela 4.6: Métricas que satisfazem o objectivo de completude da pergunta P4

P4 – Quão detalhado é o modelo de objectivos em respeito às operações?	
Nome	**PLGWOp – *P**ercentage of **L**eaf **G**oals **W**ith an **Op**eration*** (Percentagem de Objectivos Folha com uma Operação)
Definição informal	Percentagem de objectivos folha que têm, no modelo, pelo menos uma operação associada.
Definição formal	***context KAOS*** ***def: PLGWOp(): Real*** = *self.NLGWOp() / self.NLG()*
Pré-condição	*context KAOS::PLGWOp()* *pre: self.NLG() > 0*
Comentários	Se não exitem objectivos folha o resultado é indefinido.
Métricas auxiliares	
Nome	**NLGWOp – *N**umber of **L**eaf **G**oals **W**ith an **Op**eration*** (Número de Objectivos Folha com uma Operação)
Definição informal	Número de objectivos folha que têm, no modelo, pelo menos uma operação associada.
Definição formal	***context KAOS*** ***def: NLGWOp(): Integer*** = *self.hasNodes->select(n: Nodes |* *n.oclIsKindOf(Goal) and n.oclAsType(Goal).ILG() and* *n.oclAsType(Goal).GNOp() > 0)->size()*

Em [27] é definido um critério de completude dos modelos KAOS, que enuncia que *"para ser completo, um diagrama de processo deve especificar os agentes que executam as operações"*. Posto isto, na Tabela 4.7 definimos a métrica POpWA que avalia a percentagem de operações com agentes associados.

Tabela 4.7: Métricas que satisfazem o objectivo de completude da pergunta P5

P5 – Quão bem suportadas estão as operações do modelo de objectivos?	
Nome	**POpWA – *P**ercentage of **Op**erations **W**ith an **A**gent*** (Percentagem de Operações com um Agente)
Definição informal	Percentagem de operações que têm, no modelo, pelo menos um agente associado.
Definição formal	***context KAOS*** ***def: POpWA(): Real*** = *self.NOpWA() / self.NOp()*
Pré-condição	*context KAOS::POpWA()*

	pre: self.NOp() > 0
Comentários	Se não existem operações o resultado é indefinido.
Métricas auxiliares	
Nome	**NOp – *N**umber of **Op**erations*** (Número de Operações)
Definição informal	Número de operações no modelo.
Definição formal	***context KAOS*** ***def: NOp(): Integer*** = *self.hasNodes->select(n: Nodes |* *n.oclIsKindOf(OperationNode))->size()*
Nome	**NOpWA – *N**umber **Op**erations **W**ith **A**gent*** (Número de Operações com um Agente)
Definição informal	Número de operações que têm, no modelo, pelo menos um agente associado.
Definição formal	***context KAOS*** ***def: NOpWA(): Integer*** = *self.hasNodes->select(n: Nodes |* *n.oclIsKindOf(OperationNode) and* *n.oclAsType(OperationNode).OpNA() > 0)->size()*

4.3.2 Métricas para complexidade

Na Tabela 4.8 definimos métricas que ajudam a identificar se os agentes do modelo têm demasiada responsabilidade no sistema, o que pode ser sinal de má modelação. Mais especificamente, medimos a quantidade de objectivos folha associados a cada agente (visão individual), métrica ANLG, bem como um o mínimo, o máximo, e a média dos objectivos folha associados a todos os agentes do modelo (visão global).

Tabela 4.8: Métricas que satisfazem o objectivo de complexidade da pergunta P6

P6 – Será que um agente tem demasiada responsabilidade no modelo?	
Nome	**ANLG – *N**umber of **L**eaf **G**oals of an **A**gent*** (Número de Objectivos Folha com um Agente)
Definição informal	Número de objectivos folha directamente ou indirectamente associados a este agente.
Definição formal	***context Agent*** ***def: ANLG(): Integer*** = *self.ALG()->size()*
Nome	**MinNLGWA – *Min**imum **N**umber of **L**eaf **G**oals **W**ith an **A**gent*** (Número Mínimo de Objectivos Folha com um Agente)

Definição informal	Número mínimo de objectivos folha associados a um agente do modelo.

Definição formal	***context KAOS***
	def: MinNLGWA(): Integer = *self.hasNodes->select(n: Nodes |*
	n.oclIsKindOf(Agent))->iterate(n: Nodes; min: Integer = -1 |
	let aux: Integer = n.oclAsType(Agent).ANLG() in
	if min = -1 then
	aux
	else
	min.min(aux)
	endif)

Nome	**MaxNLGWA – *Max*imum *N*umber of *L*eaf *G*oals *W*ith an *A*gent** (Número Máximo de Objectivos Folha com um Agente)
Definição informal	Número máximo de objectivos folha associados a um agente do modelo.

Definição formal	***context KAOS***
	def: MaxNLGWA(): Integer = *self.hasNodes->select(n: Nodes |*
	n.oclIsKindOf(Agent))->iterate(n: Nodes; max: Integer = -1 |
	let aux: Integer = n.oclAsType(Agent).ANLG() in
	if max = -1 then
	aux
	else
	max.max(aux)
	endif)

Nome	**AveNLGWA – *Ave*rage *N*umber of *L*eaf *G*oals *W*ith an *A*gent** (Número Médio de Objectivos Folha com um Agente)
Definição informal	Número médio de objectivos folha associados a um agente do modelo.
Definição formal	***context KAOS*** ***def: AveNLGWA(): Real*** = *self.NLGWA() / self.NA()*
Pré-condição	*context KAOS::AveNLGWA()* *pre: self.NA() > 0*
Métricas auxiliares	
Nome	**NA – *N*umber of *A*gents** (Número de Agentes)
Definição informal	Número de agentes no modelo.

Definição formal	***context KAOS*** ***def: NA(): Integer*** = *self.hasNodes->select(n: Nodes |* *n.oclIsKindOf(Agent))->size()*

A Tabela 4.9 ajuda a analizar o suporte que os objectivos têm pelas ligações com os objectos do sistema. A métrica GNO avalia a complexidade segundo o número de objectos associados a um objectivo folha. Esta avaliação também é feita de forma global, através do cálculo do mínimo, máximo, e média do número de objectos associados aos objectivos folha do modelo. Estas métricas conseguem avaliar se um objectivo tem um comportamento complexo associado à manipulação de uma certa quantidade de objectos.

Tabela 4.9: Métricas que satisfazem o objectivo de complexidade da pergunta P7

P7 – Será que um objectivo folha tem muitos/poucos objectos associados?	
Nome	**GNO – *N*umber of *O*bjects of a *G*oal** (Número de Objectos de um Objectivo)
Definição informal	Número de objectos associados a este objectivos.
Definição formal	***context Goal*** ***def: GNO(): Integer*** = *self.goalConcerns->collect(concernsObject)->size()*
Nome	**MinNOWLG – *Min*imum *N*umber of *O*bjects *W*ith a *L*eaf *G*oal** (Número Minimo de Objectos com um Objectivo Folha)
Definição informal	Número mínimo de objectos associados a um objectivo folha do modelo.

Definição formal	***context KAOS***
	def: MinNOWLG(): Integer = *self.hasNodes->select(n: Nodes |*
	n.oclIsKindOf(Goal) and
	n.oclAsType(Goal).ILG())->iterate(n: Nodes; min: Integer = -1 |
	let aux: Integer = n.oclAsType(Goal).GNO() in
	if min = -1 then
	aux
	else
	min.min(aux)
	endif)

Nome	**MaxNOWLG – *Max*imum *N*umber of *O*bjects *W*ith a *L*eaf *G*oal** (Número Máximo de Objectos com um Objectivo Folha)
Definição informal	Número máximo de objectos associados a um objectivo folha do modelo.
Definição formal	***context KAOS***

	def: MaxNOWLG(): Integer = *self.hasNodes->select(n: Nodes |*
	n.oclIsKindOf(Goal) and
	n.oclAsType(Goal).ILG())->iterate(n: Nodes; max: Integer = -1 |
	let aux: Integer = n.oclAsType(Goal).GNO() in
	if max = -1 then
	aux
	else
	max.max(aux)
	endif)

Nome	**AveNOWLG – *Ave*rage *N*umber of *O*bjects *W*ith a *L*eaf *G*oal** (Número Médio de Objectos com um Objectivo Folha)
Definição informal	Número médio de objectos associados a um objectivo folha do modelo.
Definição formal	***context KAOS*** ***def: AveNOWLG(): Real*** = *self.NLGC() / self.NLG()*
Pré-condições	*context KAOS::AveNOWLG()* *pre: self.NLG() > 0*
Comentários	Se não existem objectivos folha no modelo o resultado é indefinido.
Métricas auxiliares	
Nome	**NLGC – *N*umber of *L*eaf *G*oals *C*oncerns** (Número de Preocupações do Objectivo Folha)
Definição informal	Número de precupações que os objectivos folha têm no modelo.
Definição formal	***context KAOS*** ***def: NLGC(): Integer*** = *self.hasLinks->select(l: Links |* *l.oclIsKindOf(ConcernsLink))->collect(* *oclAsType(ConcernsLink).concernedByGoal* *)->select(g: Goal | g.ILG())->size()*
Comentários	Calcula o número de conexões entre objectivos folha e objectos exitentes no modelo.

Para entender completamente um objectivo, é necessário compreender todos os objectivos pai. Assim, podemos concluir que quanto mais profunda for a hierarquia de objectivos, mais difícil é entender a racionalização dos objectivos folha. Isto pode levar a um aumento do esforço na compreensão dos modelos aquando de mudanças nos requisitos. Na Tabela 4.10 declaramos um

conjunto de métricas que descrevem a complexidade da compreensão de um objectivo folha.

Tabela 4.10: Métricas que satisfazem o objectivo de complexidade da pergunta P8

P8 – Quão difícil é de entender um objectivo folha, com respeito aos seus objectivos pai?	
Nome	**MD –** ***M****odel* ***D****epth* (Profundidade do Modelo)
Definição informal	Calcula a profundidade do modelo, tendo em conta os objectivos e os obstáculos.

Definição formal	***context KAOS***
	def: MD(): Integer =
	self.NDR(self.root.oclAsType(Nodes), Set{}, Set{0}, 0)
	->iterate(i: Integer; maxD: Integer = -1 |
	if maxD = -1 then
	i
	else
	maxD.max(i)
	endif)

Comentários	A profundidade do modelo inclui os objectivos, obstáculos, e os objectivos que sejam resoluções de obstáculos.
Métricas auxiliares	
Nome	**NDR –** ***N****odes* ***D****istance to* ***R****oot* (Distância entre Nó e Raiz)
Definição informal	Conjunto das distâncias entre cada nó do modelo, objectivo ou obstáculo, e a raiz do modelo.

Definição formal	***context KAOS***
	def: NDR(n: Nodes, visited: Set(Nodes), dist: Set(Integer), last: Integer): Set(Integer) = *if visited->one(vn: Nodes | vn = n) then*
	dist
	else
	if n.oclIsKindOf(Goal) then
	self.GDR(n.oclAsType(Goal), visited, dist, last)
	else
	if n.oclIsKindOf(Obstacle) then
	self.ODR(n.oclAsType(Obstacle), visited, dist, last)
	else

	dist *endif* *endif* *endif*
Nome	**GDR – *G*oal *D*istance to *R*oot** (Distância entre Objectivo e Raiz)
Definição informal	Conjunto das distâncias entre cada objectivo do modelo e a raiz do modelo.

Definição formal	***context KAOS***
	def: GDR(g: Goal, visited: Set(Nodes), dist: Set(Integer), last: Integer): Set(Integer) *= let visit: Set(Nodes) = visited->including(*
	g.oclAsType(Nodes)) in
	if g.ILG() then
	let obsts: Set(Obstacle) = g.goalHasObstacle
	->collect(obstacle)->asSet() in
	if obsts->notEmpty() then
	obsts->collect(o: Obstacle | NDR(o.oclAsType(Nodes),
	visit, dist->including(last+1), last+1))->asSet()
	else
	dist
	endif
	else
	union(g.goalIsAndRef->collect(andRefToOtherGoal)->union(
	g.goalIsOrRef->collect(orRefToOtherGoal)),
	g.goalHasObstacle->collect(obstacle))->collect(n: Nodes |
	NDR(n, visit, dist->including(last+1), last+1))
	->asSet()
	endif

Nome	**ODR – *O*bstacle *D*istance to *R*oot** (Distância entre Obstáculo e Raiz)
Definição informal	Conjunto das distâncias entre cada obstáculo do modelo e a raiz do modelo.
Definição formal	***context KAOS*** ***def: ODR(o: Obstacle, visited: Set(Nodes), dist: Set(Integer), last: Integer): Set(Integer)*** *= let visit: Set(Nodes) = visited->including(*

	o.oclAsType(Nodes)) in
	if o.ILO() then
	let goals: Set(Goal) = o.OS(Set{}) in
	if goals->notEmpty() then
	goals->collect(g: Goal |
	NDR(g.oclAsType(Nodes), visit, dist
	->including(last+1), last+1))->asSet()
	else
	dist
	endif
	else
	o.OR()->collect(o: Obstacle |
	NDR(o.oclAsType(Nodes), visit, dist->including(last+1),
	last+1))->asSet()
	endif

Nome	***union*** (União)
Definição informal	União entre colecções de diferentes tipos.
Definição formal	***def: union(s1: Sequence(Goal), s2: Sequence(Nodes)): Sequence(Nodes)*** = *let s: Sequence(Nodes) = Sequence{} in s->union(s1)->union(s2)*

A métrica RNSG, definida na Tabela 4.11, representa o número de sub-objectivos resultantes do refinamento do objectivo raiz do modelo. Esta é uma métrica de dimensão do modelo de objectivos como um todo, e pode ser usada como um substituto para a complexidade do mesmo. A métrica auxiliar NGSG pode ser usada para qualquer sub-nó (objectivo ou obstáculo) para ajudar a identificar problemas estruturais na decomposição dos nós. Um nó com demasiados sub-nós deve ser examinado sobre uma potencial falta de coesão.

Tabela 4.11: Métricas que satisfazem o objectivo de complexidade da pergunta P9

P9 – Quão complexo é um objectivo, com respeito aos seus refinamentos?	
Nome	**RNSG – *R**oot N**umber of S**ub-G**oals*** (Número de Sub-Objectivos da Raiz)
Definição informal	Número de sub-objectivos, directos ou indirectos, da raiz do modelo.
Definição formal	***context KAOS*** ***def: RNSG(): Integer*** = *self.root.NGSG()*

Métricas auxiliares	
Nome	**NGSG –** ***N****umber of* ***S****ub-****G****oals of a* ***G****oal* (Número de Sub-Objectivos de um Objectivo)
Definição informal	Número de sub-objectivos, directos ou indirectos, deste objectivo.
Definição formal	***context Goal*** ***def: NGSG(): Integer*** = *self.GSG(Set{})->size()*

As restantes métricas auxiliares, que não foram definidas nesta secção, podem ser encontradas no Anexo D (Tabela D.1). A Figura 4.7 apresenta a hierarquia da definição das métricas segundo o paradigma GQM.

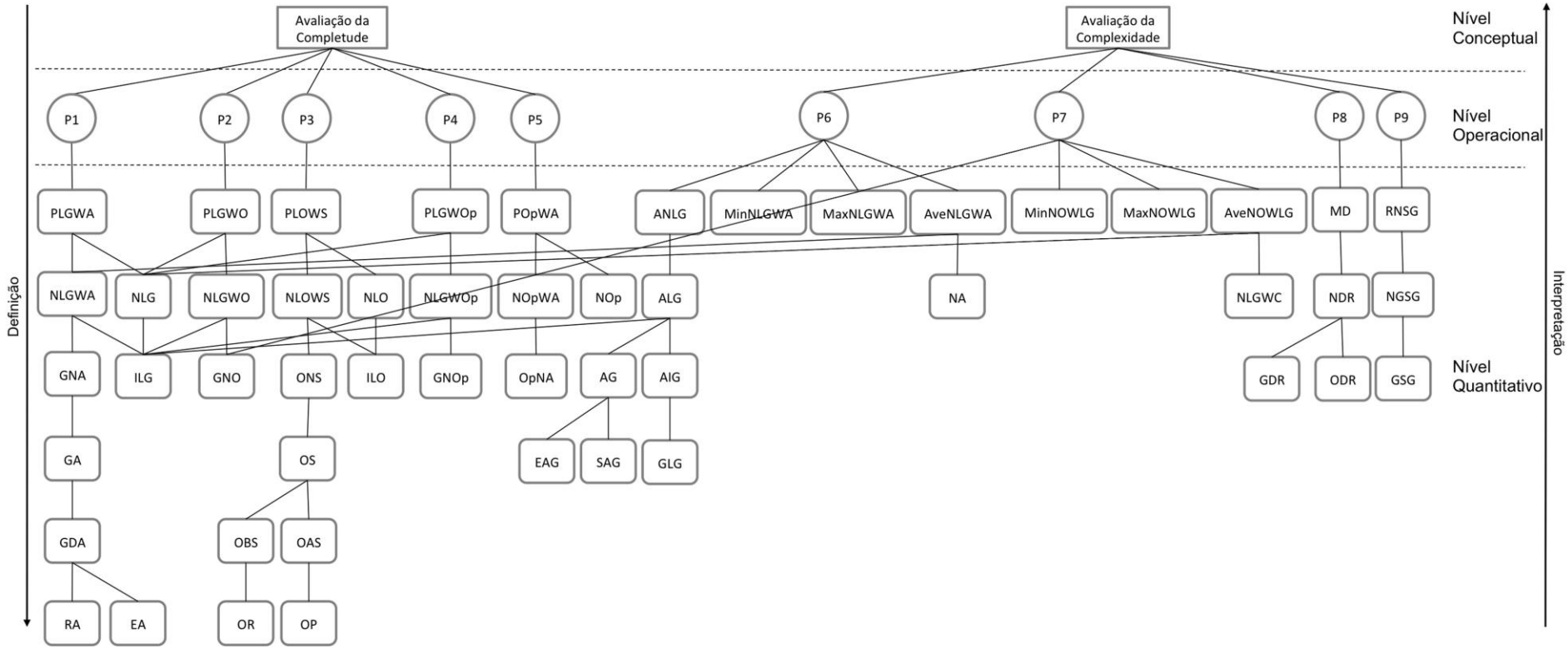


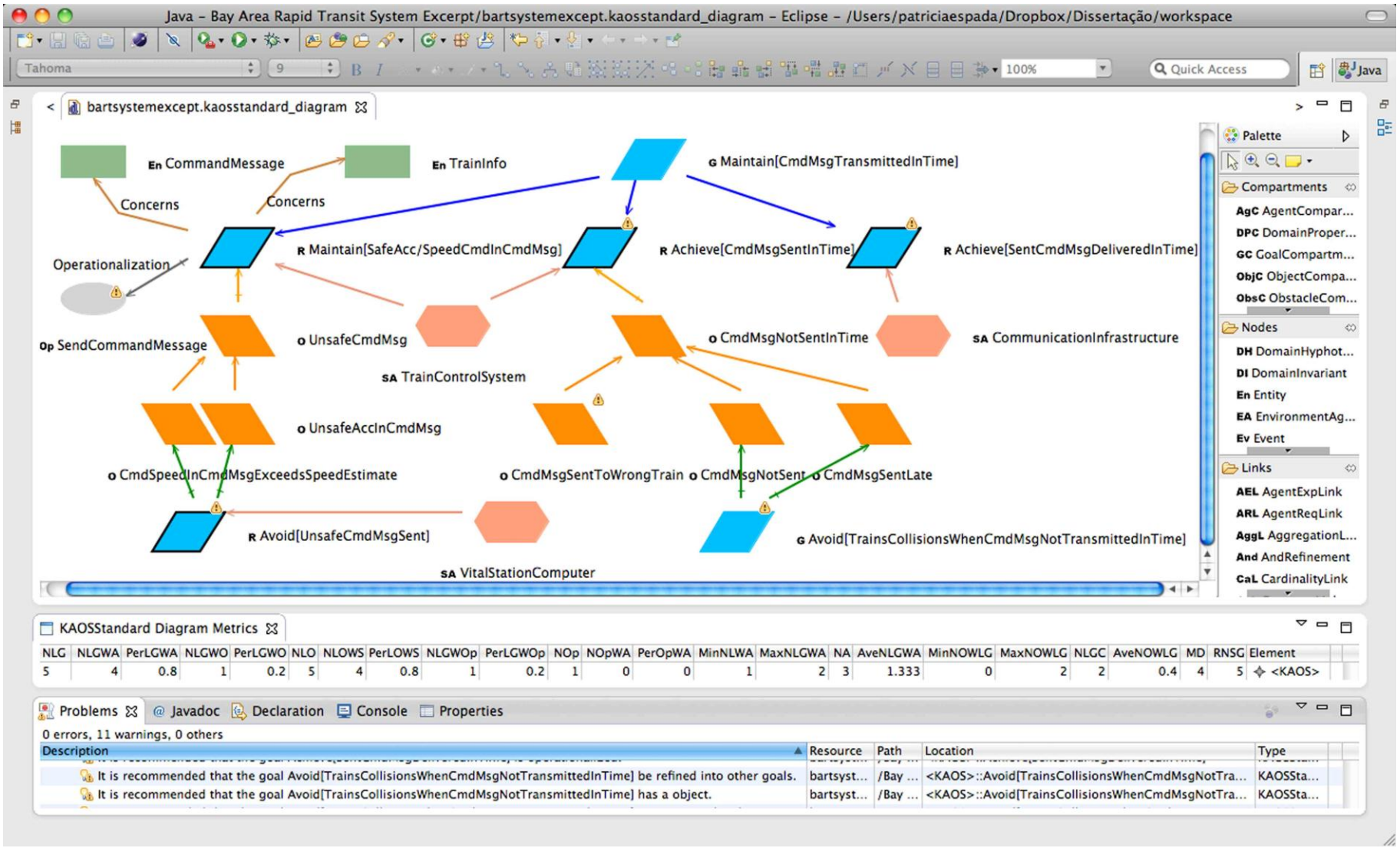
Figura 4.7: Diagrama GQM para os modelos de objectivos KAOS

4.4 Exemplo

Nesta secção vamos apresentar um excerto do caso de estudo *Bay Area Rapid Transit* (BARTS) [56], em que o principal objectivo é o de criar um sistema de comboios mais eficiente diminuindo a distância entre comboios. Isso envolve o controlo automático dos aspectos técnicos, tais como a velocidade, aceleração e distância entre os comboios. Para tal, a informação periódica de cada comboio deve ser enviada para uma estação de controlo de modo a que os comboios possam ser monitorizados e controlados em tempo real.

A Figura 4.8 mostra um fragmento do modelo BARTS modelado com a ferramenta modularKAOS. **Maintain[CmdMsgTransmittedInTime]** é um objectivo que especifica que as mensagens de comando devem ser transmitidas no tempo correcto, por forma a evitar colisões com outros comboios. Este objectivo representa o caso em que a mensagem que requer uma aceleração segura é trocada entre o comboio e o sistema de controlo, baseada na velocidade e posição dos comboios que antecedem e precedem o mesmo, e é refinado em três sub-objectivos: **Achieve[CmdMsgSentInTime]**, onde a mensagem de comando é transmitida no tempo correcto pelo **TrainControlSystem** para o comboio; **Maintain[SafeAcc/SpeedCmdInCmdMsg]**, onde o **TrainControlSystem** tenta manter uma aceleração segura nas mensagens que envia aos comboios; e **Achieve[SentCmdMsgDeliveredInTime]**, onde o agente **CommunicationInfrastructure** garante que as mensagens são enviadas. O modelo é ainda composto por alguns obstáculos e as suas respectivas resoluções. Se o obstáculo não tiver soluções, directas ou indirectas, é lançado um aviso (por exemplo, no requisito **Avoid[UnsafeCmdMsgSent]**). De igual forma, sempre que um objectivo folha não se encontra ligado a um objecto, nem a uma operação, é lançado um aviso. A operação presente no modelo lança um aviso, porque deve ser realizada por um agente e nenhum agente foi atribuído a essa operação.

Os avisos encontram-se listados na janela de problemas, debaixo da janela das métricas, que apresenta os valores da aplicação das métricas a este modelo. Neste exemplo, temos cinco objectivos folha (NLG = 5), onde quatro deles têm pelo menos um agente (NLGWA = 4), e assim por diante.



NLG	NLGWA	PerLGWA	NLGWO	PerLGWO	NLO	NLOWS	PerLOWS	NLGWOp	PerLGWOp	NOp	NOpWA	PerOpWA	MinNLWA	MaxNLGWA	NA	AveNLGWA	MinNOWLG	MaxNOWLG	NLGC	AveNOWLG	MD	RNSG	Element
5	4	0.8	1	0.2	5	4	0.8	1	0.2	1	0	0	1	2	3	1.333	0	2	2	0.4	4	5	<KAOS>

Description	Resource	Path	Location	Type
It is recommended that the goal Avoid[TrainsCollisionsWhenCmdMsgNotTransmittedInTime] be refined into other goals.	bartsyst...	/Bay ...	<KAOS>::Avoid[TrainsCollisionsWhenCmdMsgNotTra...	KAOSSta...
It is recommended that the goal Avoid[TrainsCollisionsWhenCmdMsgNotTransmittedInTime] has a object.	bartsyst...	/Bay ...	<KAOS>::Avoid[TrainsCollisionsWhenCmdMsgNotTra...	KAOSSta...

Figura 4.8: Aplicação da ferramenta modularKAOS e das métricas ao caso de estudo BARTS

4.5 Sumário

Neste capítulo vimos a aplicabilidade do GQM para a definição das métricas, e através desse paradigma definimos um conjunto de perguntas que visam avaliar os modelos de objectivos KAOS quanto à sua completude e complexidade.

O nosso interesse na ferramenta modularKAOS foi descrito com mais rigor, e foram apresentadas várias modificações, tais como: alterações ao Ecore de forma a melhor representar as nossas necessidades; integração de novas linguagens (EOL) de forma a permitir que a ferramenta seja mais amigável a futuras modificações; e validação sobre os modelos do editor gráfico não só por regras de estruturação do metamodelo, mas também através de regras OCL e EVL. Todas estas modificações permitiram tornar o processo de criação do editor gráfico mais amigável, o que não acontecia na versão do Monteiro, e também permitir a formalização das métricas em OCL sobre o metamodelo da ferramenta.

Posteriormente foi definido um conjunto de métricas que visam responder às questões propostas, bem como um vasto conjunto de métricas auxiliares que permitem todas essas medições.

5

Validação

Durante este capítulo iremos apresentar diversos casos de estudo que irão permiter uma avaliação às métricas definidas no capítulo anterior. Posto isto, para cada questão, proposta para os objectivos de completude e complexidade dos modelos de objectivos KAOS, será apresentada uma avaliação com base nos dados obtidos através da aplicação do parser Java a cada um dos casos de estudo.

5.1 Casos de estudo

Nesta secção iremos apresentar alguns casos de estudo utilizados na área de Engenharia de Requisitos Orientada a Objectivos. O nosso objectivo é aplicar as métricas formuladas na Secção 4.3 a estes casos, e a partir daí ser-nos possível fazer um julgamento com fundamento sobre a veracidade dessas métricas.

Cada caso de estudo incluirá uma breve introdução referente ao seu propósito e principais objectivos, de onde partiremos para uma apresentação do seu modelo, e respectivos resultados das métricas.

5.1.1 Sistema *Bay Area Rapid Transit*

O caso de estudo BARTS (*Bay Area Rapid Transit*) [57] preocupa-se em automatizar o sistema de controlo de comboios da área de São Francisco, de forma a servir um maior número de pessoas ao reduzir o tempo de espera entre cada comboio. Para atingir este objecto, é necessário controlar a velocidade e aceleração dos comboios no sistema e ao mesmo tempo garantir alguns requisitos de segurança, tais como: os comboios não podem entrar em carris

fechados; os comboios devem manter uma distância de segurança entre si; os comboios devem cumprir os limites de velociades. O modelo KAOS deste caso de estudo foi baseado no que Letier definiu em [56] e o resultado da sua modelação pela ferramenta modularKAOS pode ser analisado no Anexo E (Figura E.1 e Figura E.2).

5.1.2 Sistema *London Ambulance Service*

O intuito do caso de estudo LASS (*London Ambulance Service*) [58] é o de criar um sistema automatizado capaz de controlar chamadas urgentes de pedidos de ambulâncias, e, ao mesmo tempo, controlar viagens de pacientes que não sejam urgentes (estas também requerem a reserva de ambulâncias). Para garantir que o sistema está correcto e apresentar uma boa execução é importante que este seja capaz de identificar chamadas em duplicado, acidentes de maior urgência, manter-se a par da disponibilidade dos recursos (por exemplo, ambulâncias, helicópteros, entre outros), entre outras capacidades. Baseámo-nos na definição de Letier, em [56], sobre este caso de estudo, para modelarmos o modelo KAOS (ver Anexo E, Figura E.3 e Figura E.4).

5.1.3 Sistema *Elevator*

O objectivo deste caso de estudo (ES) é melhorar o desempenho e qualidade do sistema dos elevadores, garantindo que estes são seguros e que oferecem uma interface correcta (por exemplo, quando é chamado o elevador e ao premir o botão do alarme é importante que estes botões façam o pertendido). Para modelar este caso de estudo utilizámos as especificações impostas em [27]. No Anexo E, Figura E.5 e Figura E.6, encontram-se as informações resultantes da modelação deste caso de estudo pela ferramenta modularKAOS.

5.1.4 Sistema *Meeting Scheduler*

O sistema *Meeting Scheduler* (MSS) pretende suportar a organização de reuniões, incluindo, por exemplo, a definição da data e localização em que a maior parte dos participantes possam efectivamente atender à reunião [59]. O modelo deste caso de estudo pode ser consultado no Anexo E, Figura E.7 e Figura E.8, e foi baseado na definição de Lamsweerde deste caso de estudo [17].

5.1.5 Sistema *Library Management*

O caso de estudo *Library Management* (LMS) [59] prevê a implementação de um sistema que faça a gestão de bibliotecas, mais especificamente, um

sistema de bibliotecas unificado para todos os departamentos de universidade. Deve permitir o registo de utilizadores, a gestão de empréstimos, procura bibliográfica e acesso aos recursos disponíveis da biblioteca. Mais uma vez, baseámo-nos na especificação deste caso de estudo por Lamsweerde em [17]. O modelo pode ser consultado no Anexo E, Figura E.9 e Figura E.10.

5.1.6 Sistema *SMART Home*

O *SMART Home* (SHS) [60] é um caso de estudo que retrata uma casa inteligente, as suas configurações e os serviços necessários para a sua automatização. O objectivo de um sistema assim é interligar todos os equipamentos (por exemplo, luzes, termóstatos, sensores de fumo, entre outros), permitindo aos utilizadores monitorizar, configurar e controlar todos os equipamentos instalados na casa através de um controlo remoto ou um telemóvel com uma interface adequada. Baseámo-nos num trabalho feito na cadeira de Engenharia de Requisitos e Desenho de Software para modelar este sistema com a nossa ferramenta. O modelo pode ser consultado no Anexo E, Figura E.11 e Figura E.12.

5.1.7 Sistema *Mine Safety Control*

Este caso de estudo, *Mine Safety Control* (MSCS) [61], tem como objectivo aumentar a segurança dos mineiros nas minas, controlando de forma automática os níveis de água, monóxido de carbono, metano e o fluxo de ar, e também disparando o alarme sempre que estes níveis chegam a limiares críticos. O sistema deve também manter as leituras dos sensores e registar as operações das bombas, possibilitando o controlo de históricos e análise de anomalias. Baseámo-nos na definição de Lamsweerde em [17] para modelar este caso de estudo. O modelo pode ser consultado no Anexo E, Figura E.13 e Figura E.14.

5.1.8 Sistema *Car Park Management*

O propósito do caso de estudo *Car Park Management* (CPMS) é o de gerir o acesso de veículos a um parque. Para tal, é necessário administrar todos os dispositivos do parque (por exemplo, cancelas, portões, e caixas de pagamento), controlar entradas e saídas, pagamentos, entre outros. Este caso de estudo foi baseado no curso de e-Learning dado por Darimont e que pode ser consultado em [62]. Por motivos de confidencialidade não podemos apresentar o modelo deste caso de estudo.

5.2 Resultados das métricas relacionadas com o objectivo de completude

Nesta secção, para cada questão relacionada com o objectivo de completude, apresentaremos: um gráfico de colunas no lado esquerdo da imagem, onde cada coluna é referente a um caso de estudo diferente; e um gráfico *boxplot*, que mostra o grau de dispersão e assimetria dos dados. Os gráficos *boxplot* são compostos por:

- **Caixa ou *box*:** contém os valores entre o percentil 25 e o percentil 75 da amostra (sendo esta a amostra razoável – sem os valores atípicos) e apresenta três estatísticas: quartil inferior – correspondente ao limite inferior da caixa; mediana – correspondente ao centro da amostra (delimitada pelos bigodes); quartil superior – corresponde ao limite superior da caixa;
- **Bigodes ou *whiskers*:** os bigodes estendem 1,5 vezes o comprimento da caixa ou, se nenhum elemento da amostra estiver fora desses limites, indicam os valores mínimos e máximo da amostra. Numa amostra normal, cerca de 95% dos dados estão entre os bigodes;
- **Valores atípicos ou *outlier***: correspondem aos valores da amostra que se encontram desfasados dos restantes valores, e podem ser de dois tipos: atípicos – valores que se encontram à distância de uma caixa e meia do quartil inferior ou do quartil superior (representados por círculos); extremos – valores que se encontram à distância de três caixas do quartil inferior ou do quartil superior (representados por asteriscos).

P1 – PLGWA. A primeira questão preocupa-se com quão longe está o modelo de atribuir a responsabilidade de todos os seus objectivos a agentes. No lado esquerdo da Figura 5.1, podemos observar a percentagem de objectivos folha com pelo menos um agente. Essa percentagem é referente a cada modelo de objectivos de cada caso de estudo. O caso de estudo CPMS apresenta o modelo mais completo, relativamente à atribuição das responsabilidades dos objectivos, enquanto que o caso de estudo BARTS apresenta uma percentagem PLGWA muito menor em relação aos outros casos de estudo. Por esse motivo, este caso de estudo é visto como um valor atípico no gráfico *boxplot*, indicando um menor foco na atribuição de responsabilidades dos agentes do modelo. De

forma global, cerca de 70% dos objectivos folha nos modelos de objectivos cumprem a regra de completude, que especifica que um objectivo folha deve ser atribuído a um agente. Com um número par de observações (oito), não existe um valor médio único e a mediana é a média dos dois valores médios [63].

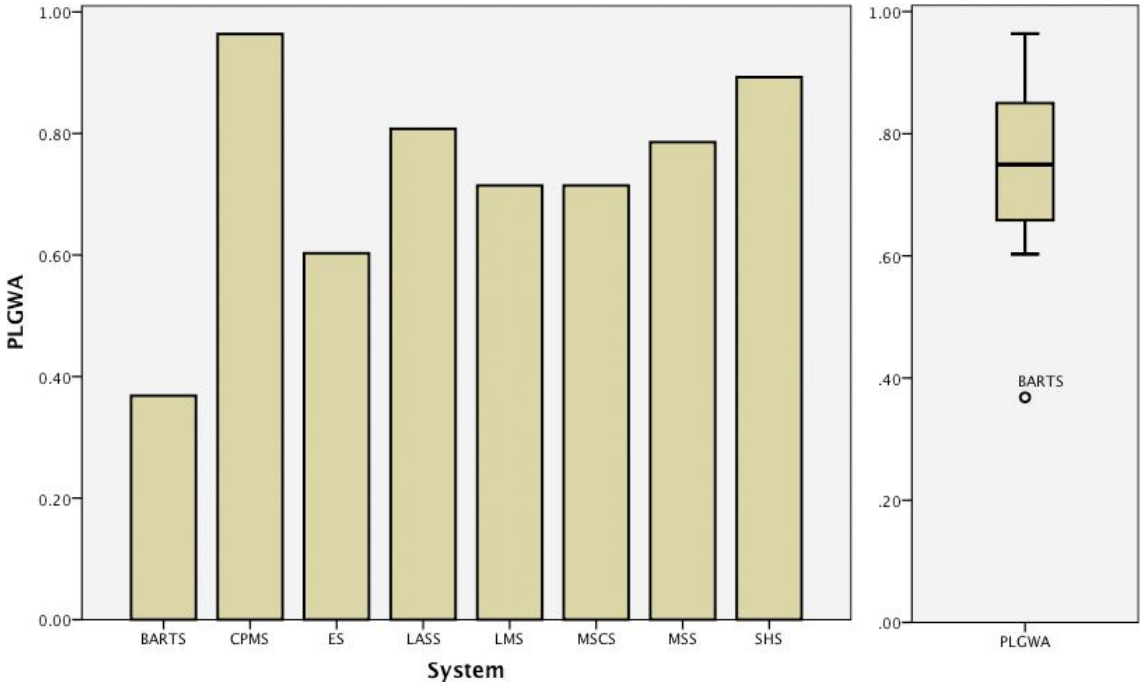


Figura 5.1: Percentagem de objectivos folha com agentes

P2 – PLGWO. A segunda questão preocupa-se com o nível de detalhe dos modelos de objectivos, com respeito à percentagem de objectivos folha associados a objectos. A Figura 5.2 mostra que a maior parte dos casos de estudo mal especificam os objectos nos seus modelos de objectivos. Os casos de estudo BARTS e LASS são os que fornecem um maior número de objectivos folha com objectos no modelo, no entanto com uma percentagem menor que 50%. Em contrapartida, os casos de estudo CPMS, LMS, MSS e SHS, negligenciam por completo a existência de objectos. No gráfico *boxplot* a mediana é abaixo de 10%, e não são identificados valores atípicos. Estes casos de estudo parecem indicar que a identificação de objectos não representa uma prioridade na fase de elicitação de requisitos.

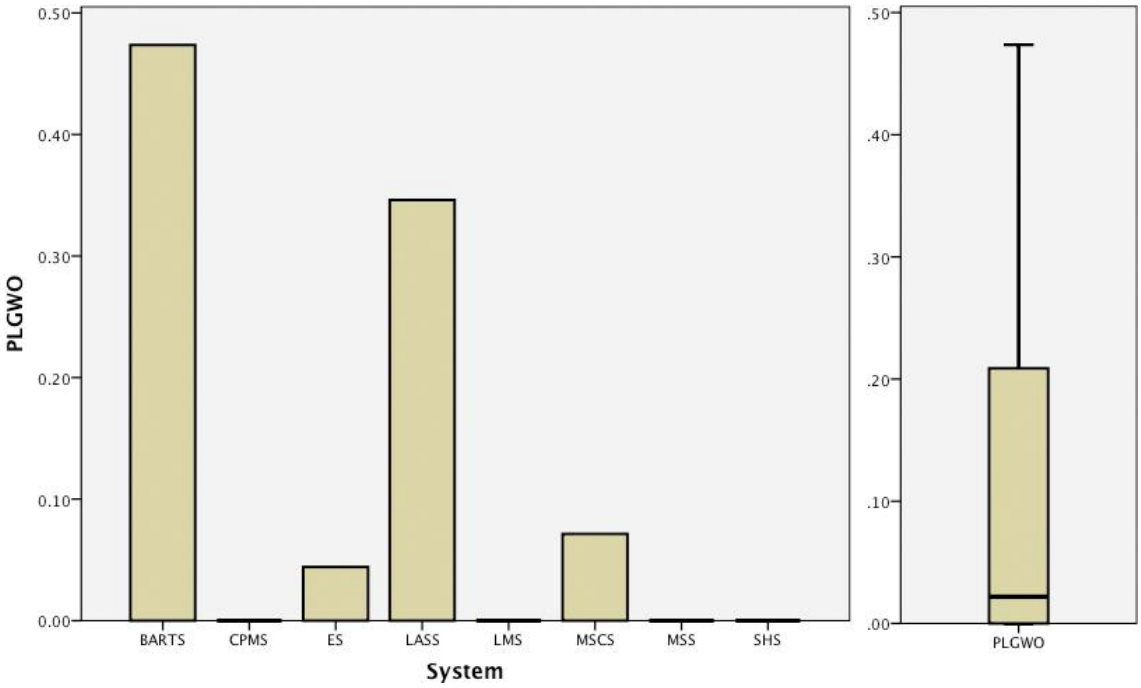


Figura 5.2: Percentagem de objectivos folha com objectos

P3 – PLOWS. A terceira questão relaciona-se com o nível de detalhe que os modelos de objectivos dão aos obstáculos com resoluções. Na Figura 5.3 podemos ver que os casos de estudo CPMS e ES, ambos retirados de tutoriais KAOS, e o caso de estudo SHS, adaptado de um trabalho realizado para uma cadeira de Engenharia de Requisitos (onde o exemplo ES é bastante referido), são aqueles que fornecem um maior número de obstáculos com resoluções. Os restantes casos de estudo têm um baixo valor de percentagem de obstáculos com resoluções, o que sugere que a especificação de resoluções para obstáculos não é uma grande preocupação nesta fase de requisitos. O caso de estudo LMS apresenta um valor nulo nesta medição por não apresentar obstáculos no seu modelo (e, por esse motivo, não apresenta as suas resoluções).

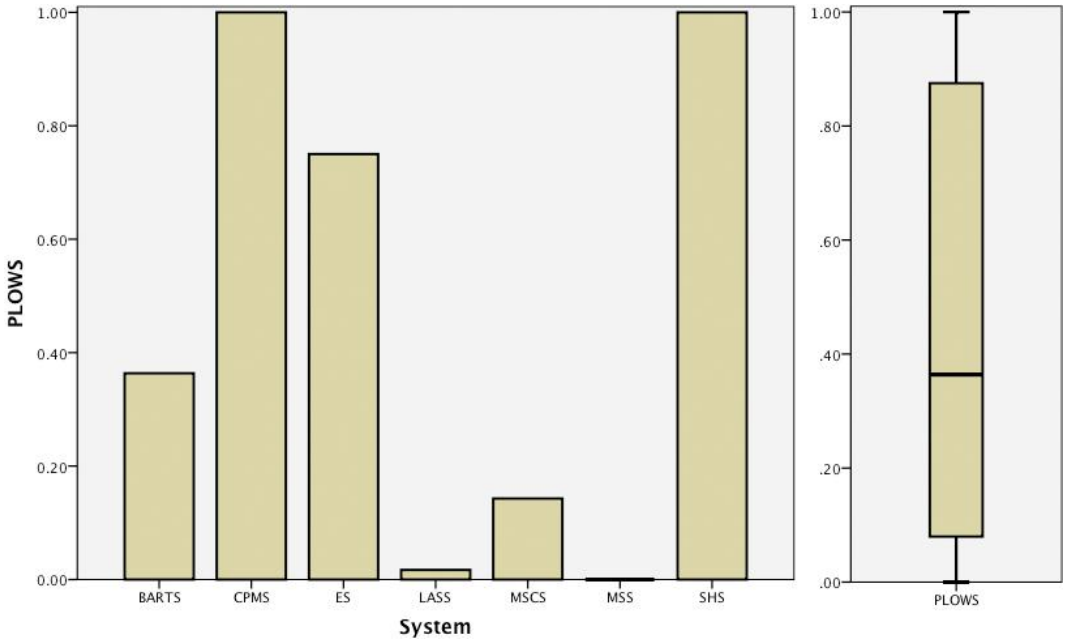


Figura 5.3: Percentagem de obstáculos folha com resoluções

P4 – PLGWOp. A quarta questão refere o nível de detalhe que o modelo de objectivos tem em relação às operações associadas aos objectivos folha. A Figura 5.4 mostra que as operações foram escassamente representadas nos casos de estudo revistos, com uma taxa de objectivos folha com operações abaixo dos 25%. MSCS é o caso de estudo onde as operações aparecem mais frequentemente. A mediana apresentada no *boxplot* é menor que 10%, mas não são identificados valores atípicos. A identificação de operações e suas associações com objectivos folha não aparenta ser uma preocupação nesta fase.

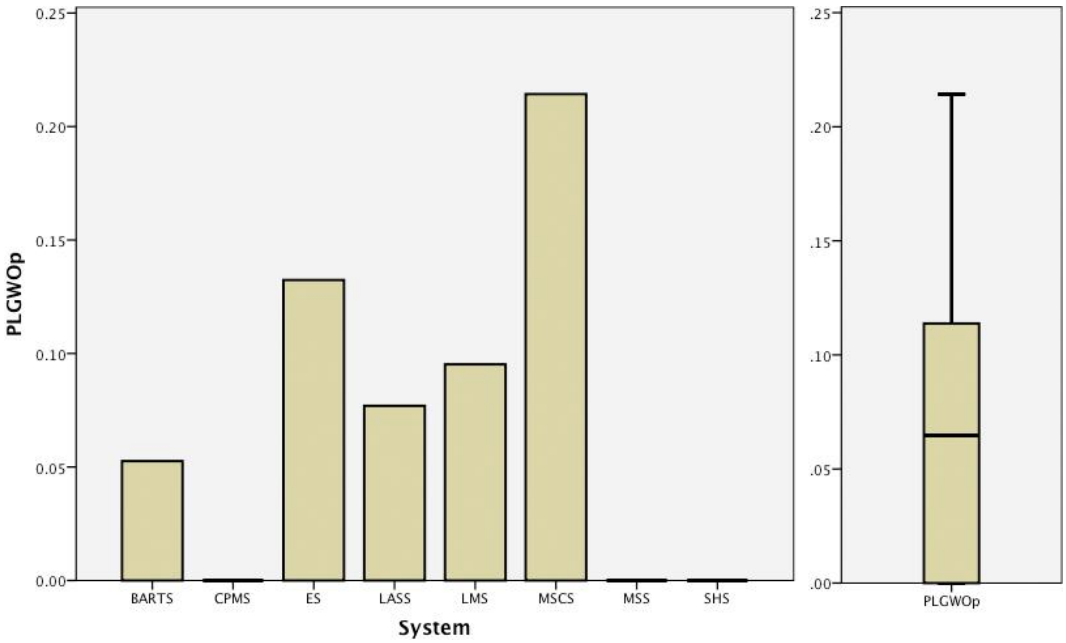


Figura 5.4: Percentagem de objectivos folha com operações

P5 – POpWA. A última questão tem como finalidade medir o nível de detalhe dos modelos de objectivos com respeito à associação de operações com agentes. É de notar, tal como pode ser visto na Figura 5.4, que os casos de estudo CPMS, MSS e SHS não apresentam qualquer operação. Por consequência, os seus valores são omitidos, tal como mostra a Figura 5.5 (não faz sentido incluir os seus valores nulos na computação da métrica POpWA, tal como indica a pré-condição na Tabela 4.7). Três dos restantes casos de estudo (BARTS, LASS e MSCS) não apresentam ligações entre operações e agentes. Portanto, restam-nos apenas dois casos de estudo, ES e LMS, onde apenas cerca de metade das operações se encontram associadas a agentes.

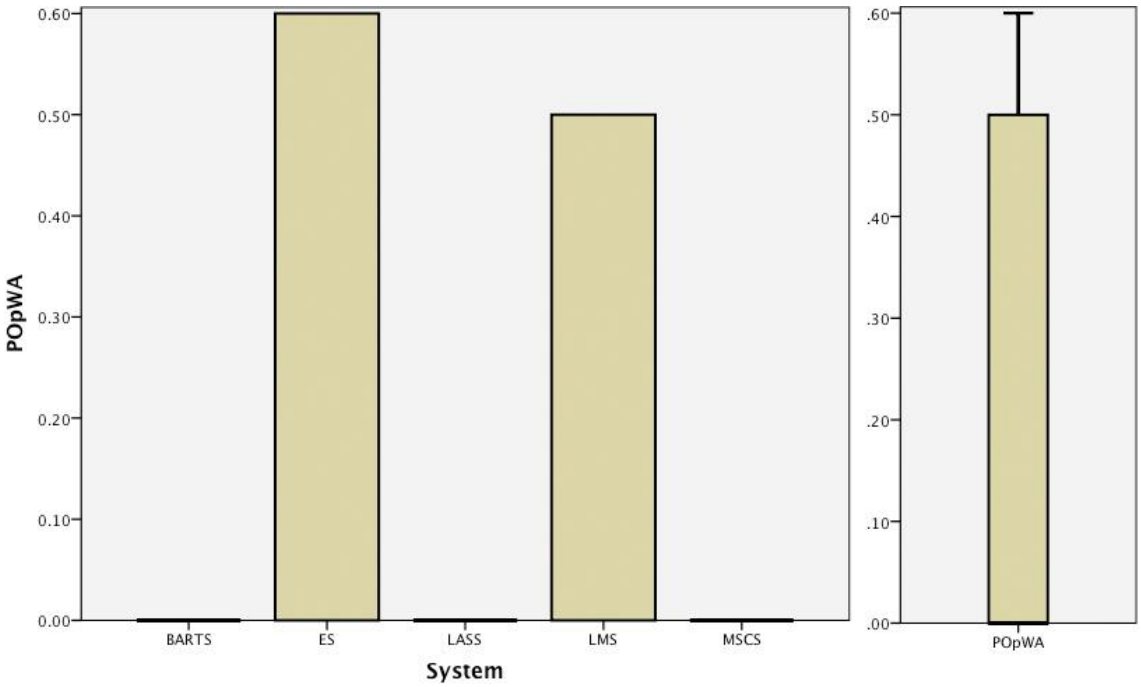


Figura 5.5: Percentagem de operações com agentes

5.3 Resultados das métricas relacionadas com o objectivo de complexidade

Para as duas primeiras questões relacionadas com a complexidade dos modelos de objectivos KAOS, apresentamos apenas o gráfico *boxplot* (visto que não estão envolvidas percentagens, apenas valores discretos). Nas últimas duas perguntas apresentamos quer os gráficos de colunas quer os *boxplots*.

P6 – ANLG. Esta questão pretende determinar o nível de responsabilidade de um único agente. Tal como mostra a Figura 5.6, a maior parte dos modelos têm cerca de 1 a 5 objectivos folha por agente. Os dois casos de estudo retirados de tutorias da metodologia KAOS (CPMS e ES) têm um número de objectivos folha por agente significantemente superior. Os casos mais extremos têm um valor ANLG muito elevado, o que sugere que o agente **CarParkController**, do caso de estudo CPMS, e o agente **Utilizador**, do caso de estudo SHS, têm demasiada responsabilidade. Estes agentes seriam possiveis candidatos para uma futura decomposição em agentes mais específicos.

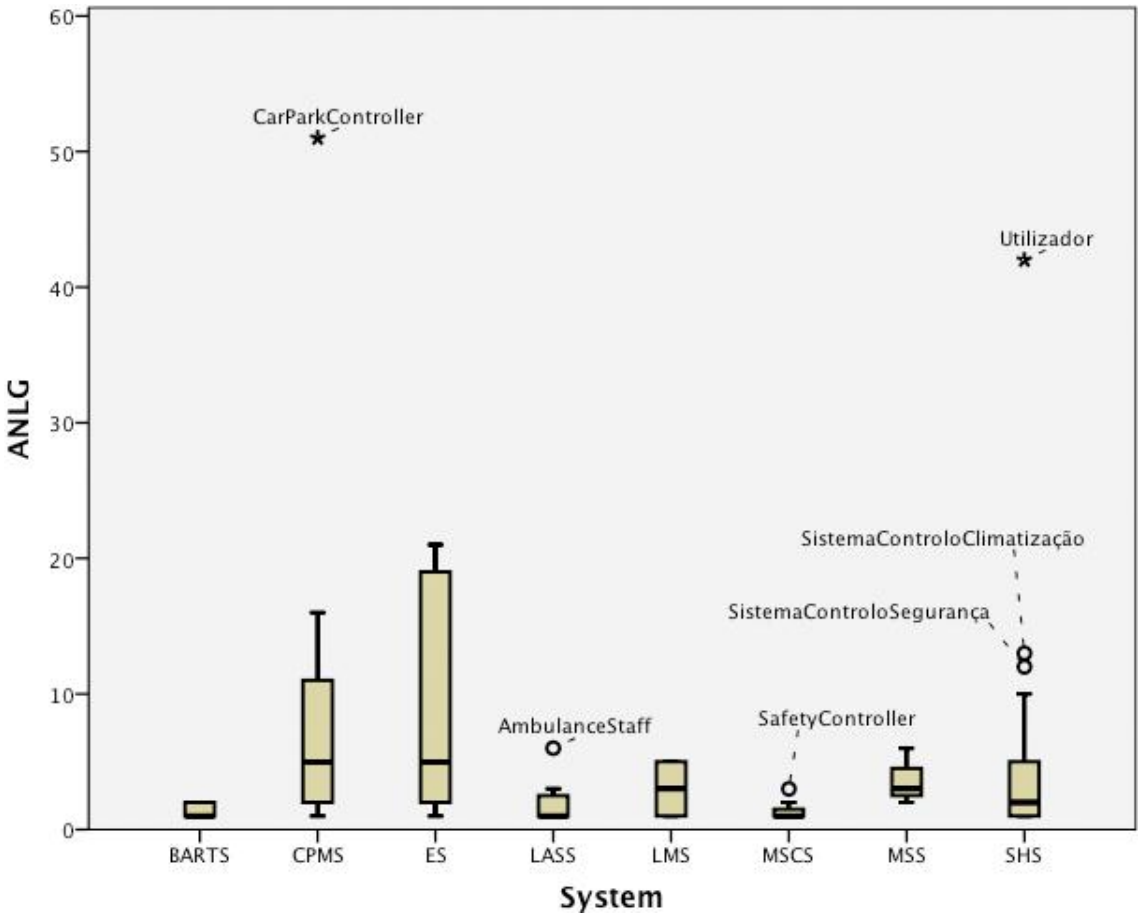


Figura 5.6: Número de objectivos folha por agente

P7 – GNO. A preocupação desta sétima questão é relativamente ao número de objectos associados a objectivos. Na Figura 5.7 mostramos como a maioria dos casos de estudo não considera os objectos nos modelos de objectivos. As excepções são os casos de estudo BARTS e LASS, onde os objectos são mais frequentemente utilizados. Os valores extremos dos casos de estudo ES e MSCS, com um objecto por objectivo, confirma o quão raramente estes elementos dos modelos são usados nos casos de estudo.

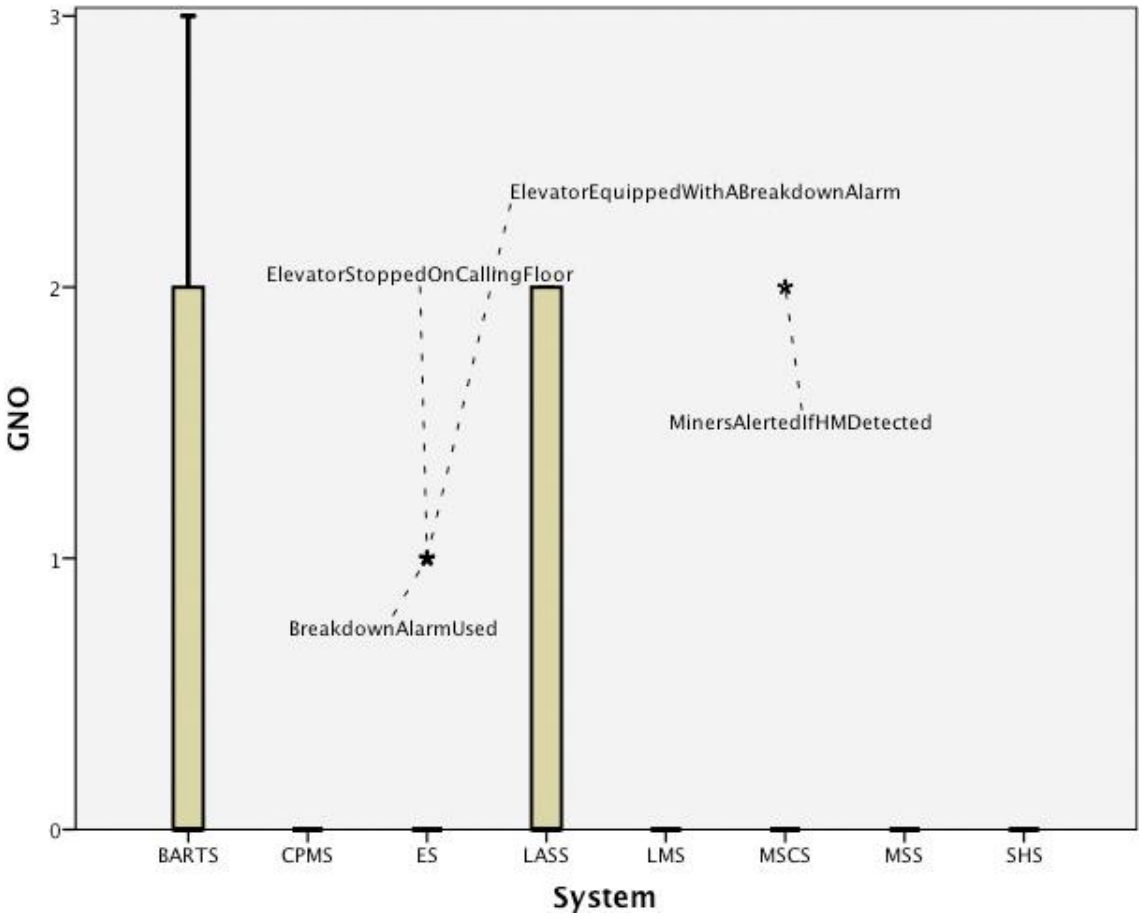


Figura 5.7: Objectos por objectivo

P8 – MD. A pergunta oito tem como objectivo medir o nível de compreensão de um modelo de objectivos, baseando-se na sua altura (utilizando uma métrica de complexidade semelhante com a proposta em [49] para o desenho orientado a objectos). Podemos observar na Figura 5.8 que na nossa amostra o sistema mais complexo é o CPMS, com 13 níveis de refinamento. No entanto, a mediana revela que o número de níveis de refinamento é cerca de 8 e nenhum valor atípico foi identificado, o que sugere que é usado um número bastante consistente de níveis de decomposição. Valores extremos nesta métrica poderiam indicar variações na complexidade acidental dos modelos. Um nível mais baixo de MD pode sugerir uma estruturação simplista do modelo, enquanto que valores muito altos podem apontar para um modelo demasiadamente refinado.

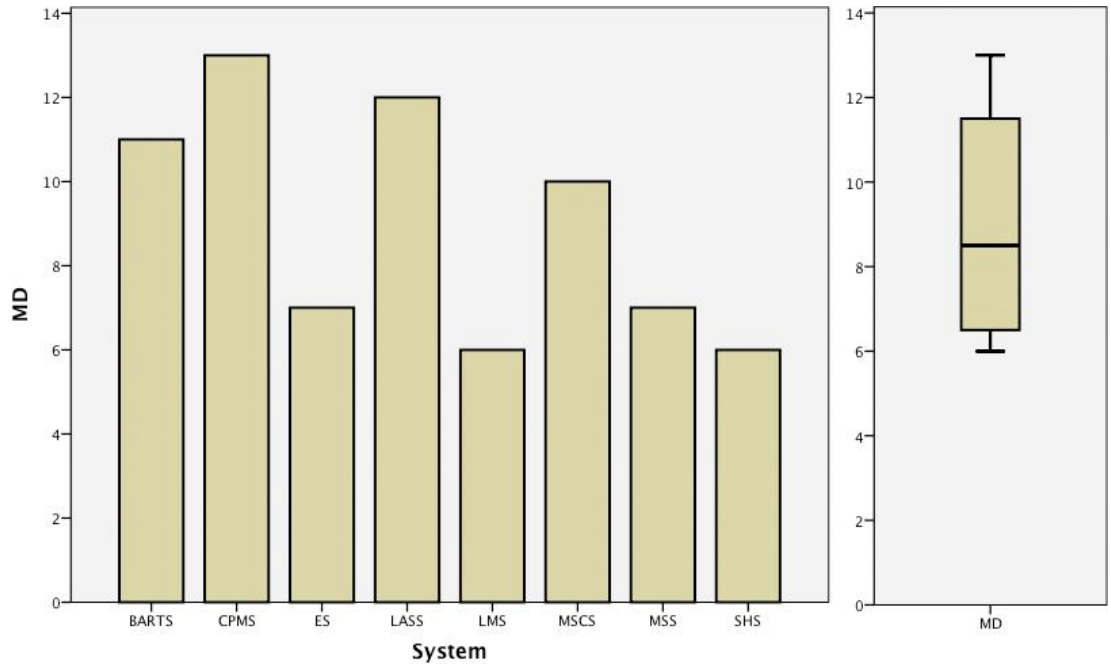


Figura 5.8: Altura do modelo

P9 – RNSG. A última pergunta é referente à avaliação da complexidade essencial, ao considerar o número de total de objectivos do modelo. A Figura 5.9 mostra que os casos de estudo CPMS e SHS tem o número mais elevado de objectivos, cerca de 200 sub-objectivos, e o gráfico *boxplot* mostra-os como fazendo parte do bigode superior da caixa, enquanto que a mediana considerando todos os casos de estudo é menor que 50. Isto sugere que estes casos de estudo sejão definidos com um maior detalhe que os restantes.

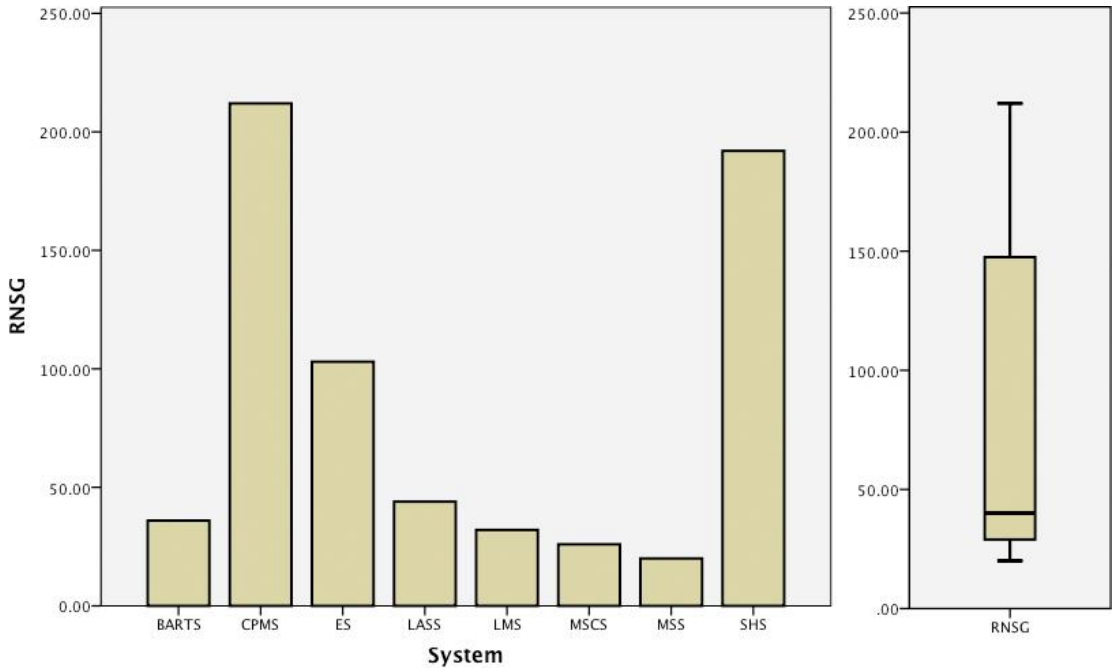


Figura 5.9: Número de sub-objectivos no modelo

5.4 Sumário

Ao longo deste capítulo analisámos vários casos de estudo, aos quais aplicámos a ferramenta modularKAOS com integração das métricas, e através dos resultados obtidos, podemos aprofundar a importância das métricas e retirar algumas observações.

Em suma, em relação ao objectivo de garantir a completude dos modelos de objectivos, observámos que na nossa amostra de modelos KAOS:

i. A atribuição de responsabilidades dos objectivos folha para os agentes é, de forma geral, uma característica considerada nos modelos;

ii. Os objectos são frequentemente esquecidos nos casos de estudo;

iii. Quando os obstáculos são especificados, encontramos uma grande variação, de 0% a 100%, da percentagem de obstáculos com resolução, sugerindo que a preocupação de especificar as resoluções dos obstáculos não é uma constante entre os engenheiros de requisitos envolvidos no desenvolvimento dos casos de estudo (alguns escolhem adiar este passo para uma fase de desenvolvimento posterior);

iv. As operações são ainda mais raras que os objectos. Em ambos os casos, parece tratar-se de uma preferência dos engenheiros de *software* de adiar a sua definição para fases do desenvolvimento do *software* posteriores;

v. Apenas dois dos casos de estudo modelam a atribuição de operações a agentes, mostrando tratar-se de um aspecto de modelação muito pouco explorado.

Quanto ao objectivo de analisar a complexidade dos modelos de objectivos KAOS, verificou-se que:

i. Na maior parte dos casos, o número de objectivos folha atribuídos a um agente é relativamente pequeno, indicando, com poucas excepções, que uma preocupação geral é a de não atribuir demasiadas responsabilidades a um único agente;

ii. Associar objectos a objectivos é um aspecto muito pouco explorado nos modelos;

iii. A altura dos modelos varia muito menos do que o número de elementos do modelo, sugerindo um estado bastante consistente da prática com relação ao que é considerado um nível de decomposição adequado ao modelo;

iv. Descobrimos grandes variações nos casos de estudo com respeito ao número de sub-objectivos definidos em cada modelo, embora o número médio seja de cerca de 40 sub-objectivos, em dois dos exemplos, é cerca de 200. Uma inspecção mais cuidada aos modelos CPMS e SHS revelou que a principal fonte de variação foi o nível significativamente superior de detalhe no qual estes modelos foram construídos.

Os casos de estudo utilizados nesta dissertação são geralmente considerados como bons exemplos de casos reais de modelos KAOS e podem ser, nesse sentido, utilizados como referência de boas práticas na modelação de objectivos. Todavia, outras especificações baseadas na indústria podem apresentar diferentes perfis de utilização dos mecanismos de modelação. No entanto, para fins de validação do conjunto de métricas proposto, a amostra de casos de estudo abrange todas as situações que pretendemos tratar com este conjunto de métricas.

6

Conclusão

Para terminar a nossa dissertação apresentamos, neste capítulo, um conjunto de ideias que permitem sublinhar a importância do nosso trabalho face ao que foi feito, e que portas foram abertas para a continuação da investigação sobre a área de avaliação de modelos orientados a objectivos.

6.1 Resumo

Nesta dissertação, propusemos, com o auxílio da abordagem GQM, um conjunto de métricas para a avaliação da completude e da complexidade dos modelos de objectivos KAOS. Esta parte do nosso trabalho foi consubstanciada no artigo [28], aceite e apresentado na conferência EmpiRE2011, que apresentava uma versão preliminar do conjunto de métricas descritas nesta dissertação. As métricas foram formalizadas utilizando a linguagem OCL, e incorporadas numa ferramenta de modelação baseada em Linguagens de Domínios Específicos (DSLs) – modularKAOS. Esta, por ter sido implementada e desenha pelos plugins EMF e GMF do IDE Eclipse, permite a integração de restrições OCL com o metamodelo da ferramenta, e também uma visão, tanto gráfica como textual, dos modelos de objectivos.

Tomando os sistemas de larga escala como contexto, o negligenciamento da completude destes sistemas nas primeiras fases de desenvolvimento do *software* pode levar a um custo inesperado nas fases de desenvolvimento posteriores. Além disso, a análise da completude é útil para ajudar os engenheiros de requisitos a verificar quão perto estão de completar o sistema. A análise da complexidade é particularmente útil para identificar problemas com a qualidade dos modelos produzidos. Em particular, pode ser usada para

ajudar a identificar oportunidades de refabricação de requisitos. A avaliação da completude e complexidade suportada pela ferramenta pode contribuir para uma melhor compreensão dos modelos de requisitos e pode ser usada para melhorar a qualidade global dos referidos modelos.

Por fim, a validação das métricas foi realizada ao aplicá-las a vários casos de estudo reais. Esses casos de estudo podem ser considerados como bons exemplos de Engenharia de Requisitos Orientada a Objectivos, visto que são utilizados não só como alvo de investigação mas também com ferramenta pedagógica, e referenciados por inúmeras fontes. Os resultados obtidos com a validação das métricas transmitem um padrão de utilização na modelação de objectivos.

6.2 Limitações

No nosso trabalho criámos uma forma de avaliar a completude e complexidade referente a alguns aspectos dos modelos de requisitos, nomeadamente: objectivos, agentes, objectos, operações, e obstáculos. No entanto, a nossa ferramenta permite uma fácil integração com outros aspectos dos modelos de requisitos, tais como, *softgoals*. Para tal, é apenas necessário descobrir um motivo de medição para esses elementos, por exemplo, para o caso da identificação de conflictos entre *softgoals* (ou objectivos no geral). Numa situação destas é de extrema importância descobrir os objectivos que entram em conflicto com os restantes, e se por acaso estamos em situação de algum objectivo só poder ser alcançado sozinho (sendo que o sistema deixa de poder satisfazer os restantes).

As medições e avaliações dos sistemas modelados são realizadas pela ferramenta modularKAOS que, apesar de interactiva e automática, pode-se tornar mais "*amiga do utilizador*". Por exemplo, permitir resoluções pré-definidas a situações irregulares, ou apresentar as métricas referentes a um elemento selecionado, seriam melhorias a introduzir na ferramenta de modo a torná-la mais aliciante.

Seria também interessante ser possível exportar os modelos do modularKAOS para outro tipo de ficheiros, como por exemplo, exportar toda a informação (dos modelos e métricas) para um ficheiro excel, onde seriam automaticamente gerados gráficos referentes às métricas. Este tipo de consulta por vezes é muito mais eficiente do que a permitida pelos modelos gráficos.

Todavia, estes dois pontos referidos, têm uma implementação facilitada já que os plugins EMF e GMF, do IDE Eclipse, são facilmente manipuláveis.

Todos estes aspectos complementam determinados atributos de qualidade associados aos modelos de objectivos de Engenharia de Requisitos Orientada a Objectivos. No nosso trabalho avaliámos a completude e complexidade, mas a compreensão e a usabilidade são também aspectos importantes.

6.3 Trabalho futuro

Após uma avaliação às métricas e aos modelos, o passo seguinte será propor formas de identificar oportunidades de melhoria dos modelos, com base nas métricas propostas. Para isso, seria interessante pensar numa solução automática de integração das métricas, com catálogos de padrões e soluções de refabricação. Contudo, é necessário aplicar as métricas a sistemas da indústria, para comprovar os comportamentos analisados nesta dissertação. Ao comprovar que tanto sistemas académicos como sistemas ligados à industria aparentam ter um comportamento semelhante, será mais fácil de traçar problemas comuns e suas soluções.

Pretendemos também, como trabalho futuro, estender o conjunto de métricas definido para cobrir outros atributos de qualidade do modelo e replicar essa avaliação com outros modelos KAOS. Este seria um passo em direcção à integração de heurísticas de modelação baseadas em métricas com ferramentas de Engenharia de Requisitos Orientada a Objectivos.

Bibliografia

[1] A. Dardenne, A. v. Lamsweerde, S. Fickas, "Goal-Directed Requirements Acquisition", Sixth International Workshop on Software Specification and Design, Elsevier Science Publishers B. V., 1993, pp. 3-50, doi.

[2] A. v. Lamsweerde, E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering", IEEE Transactions on Software Engineering, vol. 26, No. 10, pp. 978-1005, October, 2000, doi: 10.1109/32.879820.

[3] A. Lapouchnian, "Goal-Oriented Requirements Engineering: an Overview of the Current Research," Technical report, University of Toronto, Toronto, Canadá, 2005.

[4] E. Yu, "Social Modeling and i*", Conceptual Modeling: Foundations and Applications. vol. 5600, A. Borgida, *et al.*: Springer Berlin / Heidelberg, 2009.

[5] J. Mylopoulos, L. Chung, B. Nixon, "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach", IEEE Transactions on Software Engineering, vol. 18, No. 6, pp. 483-497, 1992, doi: 10.1109/32.142871.

[6] A. Abran, J. W. Moore, P. Bourque, R. Dupuis, L. L. Tripp, *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, EUA: IEEE Computer Society, 2004.

[7] G. Kotonya, I. Sommerville, "Requirements Engineering: Processes and Techniques", Worldwide series in computer science, pp. xi, 282 p., 1998.

[8] A. v. Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", 5th IEEE International Symposium on Requirements Engineering, IEEE Computer Society, Toronto, Canadá, 2001, pp. 249-262, doi.

[9] G. G. Schulmeyer, "Handbook of Software Quality Assurance", pp. xx, 464 p., 2008.

[10] IEEE. Accessed on June, 2011. IEEE Standard for a Software Quality Metrics Methodology. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=749159

[11] A. Dias, "Uma Linguagem específica do domínio para uma abordagem orientada aos objectivos baseada em KAOS", Tese de Mestrado, Departamento de Informática, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, Lisboa, Portugal, 2009.

[12] R. Monteiro, "Engenharia de Requisitos Orientada a Modelos para Abordagens Orientadas a Objectivos", Tese de Mestrado, Departamento de Informática, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, Lisboa, Portugal, 2010.

[13] I. Sommerville, *Software Engineering*, 9 ed.: Addison-Wesley, 2010.

[14] I. Sommerville, "Why software engineering?", Software Engineering for Microprocessor Systems, P. Depledge, London: Peregrinus on behalf of the Institution of Electrical Engineers, 1983.

[15] W. W. Royce, "Managing the Development of Large Software Systems", in IEEE Wescon, 1970.

[16] B. W. Boehm, "A Spiral Model of Software Development and Enhancement", SIGSOFT Software Engineering Notes, vol. 11, No. 4, 1986, doi: 10.1145/12944.12948.

[17] A. v. Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Hoboken, EUA: John Wiley & Sons, Inc., ISBN: 9780470012703, 2009.

[18] P. Zave, "Classification of Research Efforts in Requirements Engineering", Second IEEE International Symposium on Requirements Engineering (RE'95), IEEE Computer Society Washington, DC, EUA, 1995, pp. 214-, doi.

[19] B. Nuseibeh, S. Easterbrook, "Requirements Engineering: a Roadmap", Conference on The Future of Software Engineering (ICSE '00), ACM, Limerick, Irlanda, 2000, pp. 35-46, doi: 10.1145/336512.336523.

[20] A. v. Lamsweerde, "Requirements Engineering in the Year 00: a Research Perspective", 22nd International Conference on Software Engineering (ICSE'00), ACM, Limerick, Irlanda, 2000, 10.1109/ICSE.2000.870392

[21] G. Kotonya, I. Sommerville, *Requirements Engineering: Processes and Techniques*. Nova Iorque, EUA: John Wiley & Sons, Inc., ISBN: 0471972088 (hbk. alk. paper), 1998.

[22] A. T. Bahill, F. F. Dean, "Discovering System Requirements", Handbook of Systems Engineering and Management (2nd Edition), A. P. Sage and W. B. Rouse, 2 ed: John Wiley & Sons, Inc., 2009.

[23] J. Araújo, E. Baniassad, P. Clements, A. Moreira, A. Rashid, B. Tekinerdogan, "Early Aspects: the Current Landscape," Technical report, Lancaster University, 2005.

[24] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Conallen, K. Houston, *Object-Oriented Analysis and Design with Applications (3rd Edition)*, 3rd ed.: Addison-Wesley Professional, ISBN: 9780201895513, 2007.

[25] OMG. Accessed on June, 2011. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. Available: http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF

[26] V. M. Werneck, A. Oliveira, J. C. Leite, "Comparing GORE Frameworks: i-star and KAOS", Ibero-American Workshop of Engineering of Requirements, Valparaíso, Chile, 2009.

[27] Respect-IT, "A KAOS Tutorial, version 1.0", October, 2007.

[28] P. Espada, M. Goulão, J. Araújo, "Measuring Complexity and Completeness of KAOS Goal Models", International Workshop on Empirical Requirements Engineering, EmpiRE 2011, IEEE Computer Society, Trento, Itália, Agosto, 2011, pp. 29-32, doi: 10.1109/EmpiRE.2011.6046252.

[29] Respect-IT. Accessed on February, 2012. Objectiver. Available: http://www.objectiver.com/

[30] E. Kavakli, P. Loucopoulos, "Goal Driven Requirements Engineering: Evaluation of Current Methods", in the 8th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD '03), Velden, Áustria, 2003.

[31] R. v. Solingen, E. Berghout, "Integrating Goal-Oriented Measurement in Industrial Software Engineering: Industrial Experiences with and Additions to the Goal/Question/Metric Method (GQM)", the 7th International Symposium on Software Metrics (METRICS '01), IEEE Computer Society, 2001.

[32] Accessed on April 2011. Victor R. Basili. Available: http://www.computer.org/portal/web/awards/basili

[33] Accessed on April 2011. Victor R. Basili's Publications. Available: http://www.cs.umd.edu/~basili/papers.html

[34] R. v. Solingen, E. Berghout, *The Goal/Question/Metric Method: a Practical Guide for Quality Improvement of Software Development*. England: McGraw Hill, ISBN: 0077095537, 1999.

[35] V. R. Basili, G. Caldiera, H. D. Rombach, "The Goal Question Metric Approach", Encyclopedia of Software Engineering, J. J. Marciniak: John Wiley & Sons, Inc., 1994, pp. 528-532.

[36] J. Esteves, J. Pastor, J. Casanovas, "Measuring Sustained Management Support in ERP Implementation Projects: a GQM Approach", in 8th Americas Conference on Information Systems (AMCIS), Dallas, USA, 2002, pp. 1381-1389.

[37] T. Kelly, R. Weaver, "The Goal Structuring Notation - A Safety Argument Notation", in Dependable Systems and Networks 2004 Workshop on Assurance Cases (DSN-2004), Florença, Itália, 2004.

[38] T. Kelly, "Arguing Safety - A Systematic Approach to Managing Safety Cases", PhD Thesis, Department of Computer Science, University of York, York, Reino Unido, 1998.

[39] R. Ramos, J. Castro, J. Araújo, A. Moreira, F. Alencar, E. Santos, R. Penteado, "AIRDoc–An Approach to Improve Requirements Documents", in 22th Brazilian Symposium on Software Engineering (SBES'08), Brasil, 2008.

[40] R. Ramos, J. Castro, J. Araújo, F. Alencar, "Towards the improvement of use case models: the AIRDoc process", ACM Symposium on Applied Computing, ACM, TaiChung, Taiwan, 2011, pp. 708-709, doi: 10.1145/1982185.1982339.

[41] G. Giachetti, F. Alencar, X. Franch, O. Pastor, "Applying *i** Metrics for the Integration of Goal-Oriented Modelling into MDD Processes," Technical report, Universitat Politècnica de Catalunya, Barcelona, Espanha, 2010.

[42] C. P. Ayala, C. Cares, J. P. Carvallo, G. Grau, M. Haya, G. Salazar, X. Franch, E. Mayol, C. Quer, "A Comparative Analysis of i*-Based Agent-Oriented Modeling Languages", in 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05), Taipei, Taiwan, República da China, 2005.

[43] M. Lucena, E. Santos, C. Silva, F. Alencar, M. J. Silva, J. Castro, "Towards a Unified Metamodel for i*", 2nd IEEE International Conference On Research Challenges in Information Science (RCIS 2008), 2008, pp. 237-246, doi.

[44] X. Franch, G. Grau, "Towards a Catalogue of Patterns for Defining Metrics over i* Models", 20th International Conference on Advanced Information Systems Engineering (CAiSE '08), Springer-Verlag, Montpellier, França, 2008, pp. 197-212, doi: 10.1007/978-3-540-69534-9_16.

[45] G. Giachetti, B. Marín, O. Pastor, "Using UML as a Domain-Specific Modeling Language: A Proposal for Automatic Generation of UML Profiles", 21st International Conference on Advanced Information Systems Engineering, Springer-Verlag, Amsterdão, Holanda, 2009, pp. 110-124, doi: 10.1007/978-3-642-02144-2_13.

[46] X. Franch, "A Method for the Definition of Metrics over i* Models", 21st International Conference on Advanced Information Systems Engineering, Springer-Verlag, Amsterdão, Holanda, 2009, pp. 201-215, doi: 10.1007/978-3-642-02144-2_19.

[47] J. Pimentel, X. Franch, J. Castro, "Measuring Architectural Adaptability in *i** Models", in XIV Ibero-American Conference on Software Engineering (CIbSE 2011), Rio de Janeiro, Brasil, 2011.

[48] C. Cares, X. Franch, "Towards a Framework for Improving Goal-Oriented Requirement Models Quality", in 12th Workshop on Requirements Engineering, Valparaiso, Chile, 2009.

[49] S. R. Chidamber, C. F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, No. 6, pp. 476-493, June, 1994, doi: 10.1109/32.295895.

[50] Eclipse. Accessed on February, 2012. EMF, Eclipse Modeling Framework Project. Available: http://eclipse.org/modeling/emf/

[51] Eclipse. Accessed on February, 2012. GMF, Eclipse Graphical Modeling Framework. Available: http://www.eclipse.org/modeling/gmp/

[52] Eclipse. Accessed on March, 2012. OCLInEcore. Available: http://wiki.eclipse.org/MDT/OCLinEcore

[53] Eclipse. Accessed on February, 2012. EMFatic, Eclipse Modeling Framework Technology. Available: http://wiki.eclipse.org/Emfatic

[54] Eclipse. Accessed on March, 2012. EOL, Epsilon Object Language. Available: http://www.eclipse.org/gmt/epsilon/doc/eol/

[55] Eclipse. Accessed on February, 2012. EVL, Epsilon Validation Language. Available: http://www.eclipse.org/epsilon/doc/evl/

[56] E. Letier, "Reasoning about Agents in Goal-Oriented Requirements Engineering", PhD Thesis, Department of Computer Engineering, Catholic University of Louvain, Louvain, 2001.

[57] V. L. Winter, R. S. Berg, J. T. Ringland, "Bay area rapid transit district advance automated train control system case study description", High integrity software: Kluwer Academic Publishers, 2001, pp. 115-135.

[58] A. Finkelstein, "The London Ambulance System Case Study", in 8th Intl. Workshop on Software Specification and Design (IWSSD 8), 1996, pp. 5-19.

[59] M. S. Feather, S. Fickas, A. Finkelstein, A. V. Lamsweerde, "Requirements and Specification Exemplars", Automated Software Engg., vol. 4, No. 4, pp. 419-438, 1997, doi: 10.1023/a:1008680612960.

[60] R. Harper, *Inside the smart home*: Springer, ISBN: 9781852336882, 2003.

[61] A. Burns, A. M. Lister, "A framework for building dependable systems", Comput. J., vol. 34, No. 2, pp. 173-181, 1991, doi: 10.1093/comjnl/34.2.173.

[62] Respect-IT. Accessed on February, 2012. e-Learn GORE & Objectiver. Available: http://www.objectiver.com/index.php?id=108

[63] E. W. Weisstein. Accessed on February, 2012. Statistical Median - MathWorld--A Wolfram Web Resource. Available: http://mathworld.wolfram.com/StatisticalMedian.html

A

Metamodelo do modularKAOS

Como já foi referido na Secção 2.3.1 a ferramenta modularKAOS permite a representação dos seguintes modelos KAOS: objectivos, responsabilidades e objectos. O metamodelo (Figura A.1) desta ferramenta contém o nó raiz numa classe denominada *KAOS*. Esta classe encontra-se dividida em três classes abstractas: *Nodes*, *Links*, e *CompartmentNode*.

A classe *Nodes*, que representa os nós, tem como função agrupar os conceitos suportados pela ferramenta, estando esses especificados nas seguintes classes:

- *DomainProperties*, classe abstracta que visa representar as propriedades de domínio podendo estas ser invariantes (*DomainInvariant*) ou (*DomainHyphothesis*);
- *OperationNode*, classe que representa operações;
- *Goal*, classe que representa objectivos, podendo estes ser expectativas (*Expectations*), requisitos (*Requirements*) ou *Softgoals*;
- *Obstacle*, classe que representa obstáculos;
- *Object*, classe abstracta que representa relações (*Relationships*), entidades (*Entity*), eventos (*Event*) e agentes (*Agent*), que por sua vez podem ser agentes do sistema (*SystemAgent*) ou agentes do ambientes (*EnvironmentAgent*).

Para construir os modelos não basta apresentar um conjunto de termos, é necessário definir como estes se interligam. Daqui surge a importância da classe

abstracta *Links*, que guarda os vários tipos de ligações que podem existir entre os diferentes nós, nomeadamente:

- *AndRefinement*, ligação de refinamento-E entre dois nós do tipo *Goal*. O refinamento é feito sobre o nó mais abstracto para um nó mais específico.
- *OrRefinement*, ligação de refinamento-OU entre dois nós do tipo *Goal*. O refinamento é feito sobre o nó mais abstracto para um nó mais específico.
- *AgentReqLink*, ligação de responsabilidade de um agente do tipo *SystemAgent* para um objectivo do tipo *Requirement*. Esta ligação representa a responsabilidade de uma agente sobre um objectivo.
- *AgentExpLink*, ligação de atribuição de um objectivo do tipo *Expectation* para um agente do tipo *EnvironmentAgent*. Esta ligação representa a atribuição de um objectivo a um agente.
- *ObstructionLink*, ligação de obstrução entre um nó do tipo *Goal* e um do tipo *Obstacle*. Significa que o ponto de falha do objectivo é o obstáculo indicado.
- *SolutionLink*, ligação de solução de uma obstrução entre um nó do tipo *Obstacle* e um do tipo *Goal*. Significa que a solução para o obstáculo passa por garantir o objectivo.
- *ConcernsLink*, ligação de preocupação entre um nó do tipo *Goal* e um do tipo *Entity*. Significa que o objectivo é da preocupação da entidade.
- *DomainPropLink*, ligação de propriedade do domínio entre um nó do tipo *Goal* e um do tipo *DomainProperty*. A propriedade do domínio é declarada pelo objectivo indicando que este satisfaz essa mesma propriedade.
- *MonitorsLink*, ligação de monitorização entre um nó do tipo *Agent* e um do tipo *Object*. Significa que o agente monitoriza o objecto.
- *ControlsLink*, ligação de controlo entre um nó do tipo *Agent* e um do tipo *Object*. Significa que o agente controla o objecto.

- *OperationalizationLink*, ligação de operacionalização entre um nó do tipo *Goal* e um do tipo *OperationNode*. Esta ligação torna o objectivo operacional.
- *CardinalityLink*, ligação de cardinalidade entre dois nós do tipo *Object*.
- *InheritanceLink*, ligação de herança entre dois nós do tipo *Object*. A herança é do primeiro nó para o segundo, sendo que o primeiro é que herda as propriedades do segundo.
- *AggregationLink*, ligação de agregação entre dois nós do tipo *Object*. A agregação é feita do segundo nó para o primeiro, ou seja, o primeiro nó tem uma ligação de um para muitos com o segundo nó.
- *ObstacleRefinement*, ligação de refinamento entre dois nós do tipo *Obstacle*. O refinamento é feito do nós mais abstracto para o nó mais específico.

A classe abstracta *CompartmentNode* tem a função de agrupar elementos que representam compartimentos através de relações de herança. Um compartimento é um contentor para elementos de diagramas KAOS introduzidos para ajudar a minimizar os efeitos de escalabilidade nos modelos de Engenharia de Requisitos Orientada a Objectivos. Podem ser de vários tipos: *GoalCompartmentNode*; *SoftgoalCompartmentNode*; *ObstacleCompartmentNode*; *DomainPropertiesCompartmentNode*; *AgentCompartmentNode*; *ObjectCompartmentNode*; *OperationCompartmentNode*.

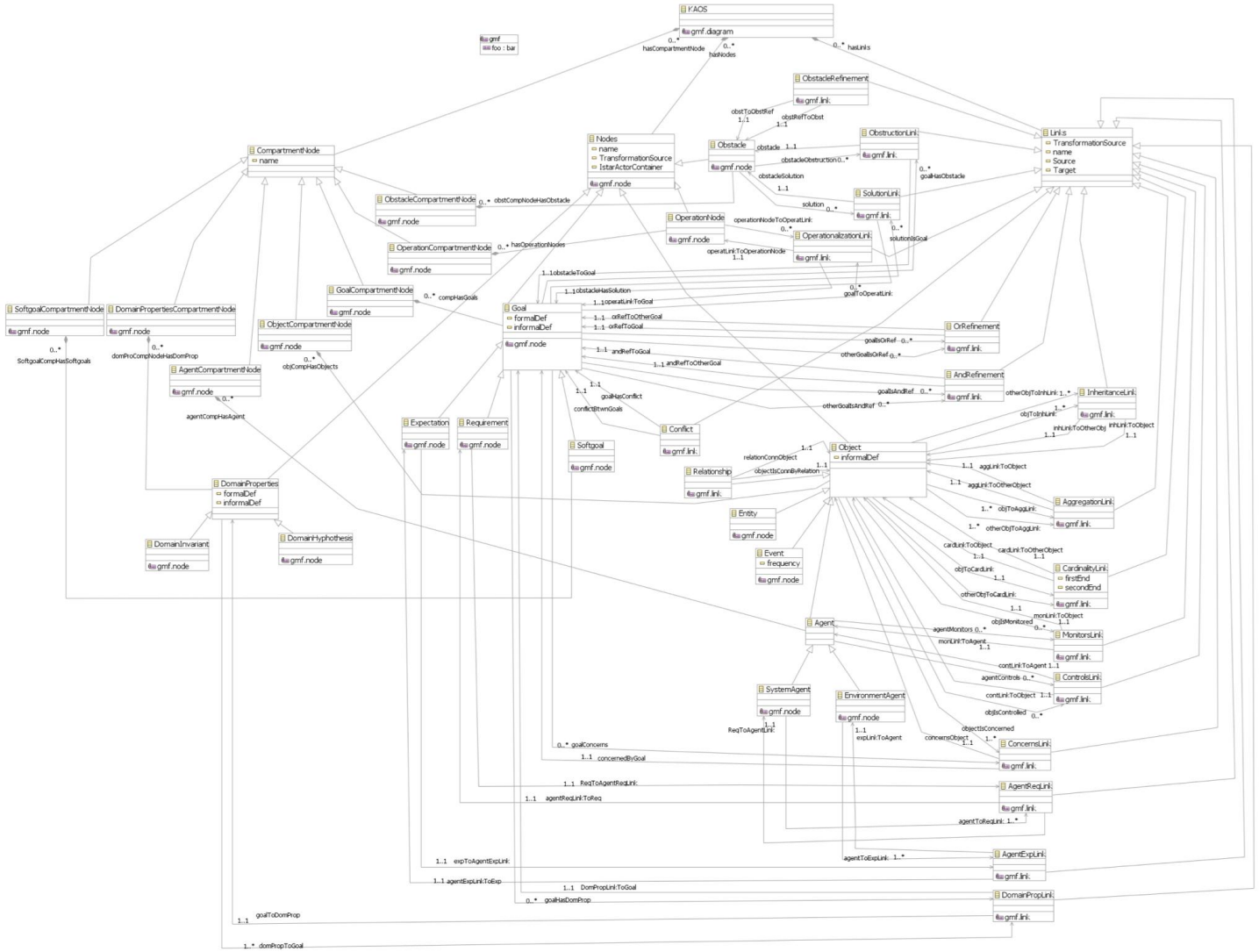


Figura A.1: Metamodelo do modularKAOS (versão Rui Monteiro) [12]

B

Nova versão do metamodelo do modularKAOS

A Figura B.1 apresenta o metamodelo da ferramenta modularKAOS contemplando todas as modificações declaradas na Secção 4.2.2 e que foram necessárias para o bom funcionamento e integração com especificação de métricas por meio da linguagem OCL.

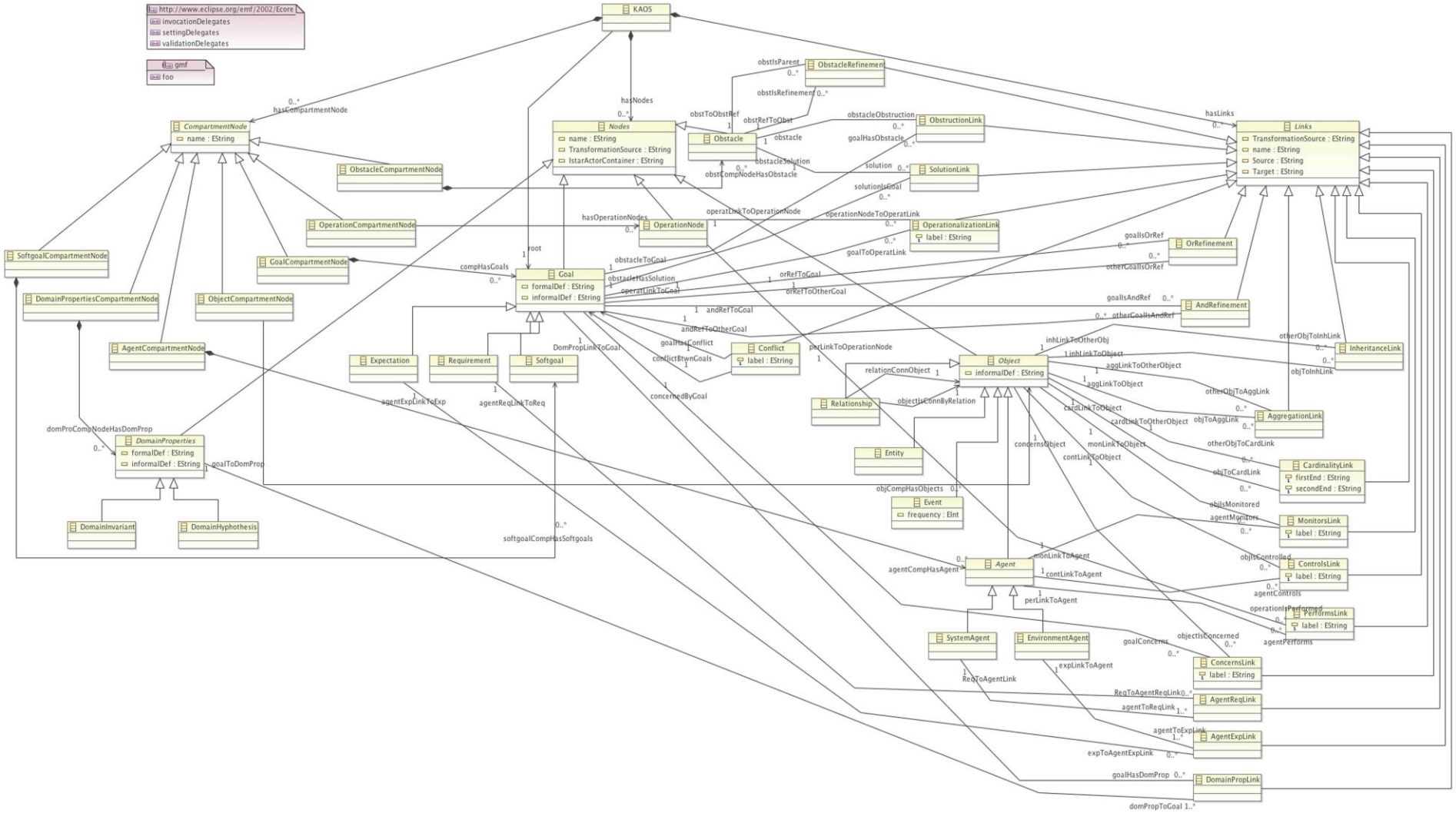


Figura B.1: Metamodelo do modularKAOS (nova versão)

C

modularKAOS - Manual do utilizador

C.1 Requisitos do sistema

A ferramenta modularKAOS foi desenvolvida sobre o Eclipse Modeling Tools, versão Juno 20120202-1513. A Figura C.1 mostra os plugins e respectivas versões que utilizamos para o desenvolvimento do modularKAOS, e sobre estes podemos garantir o seu bom funcionamento.

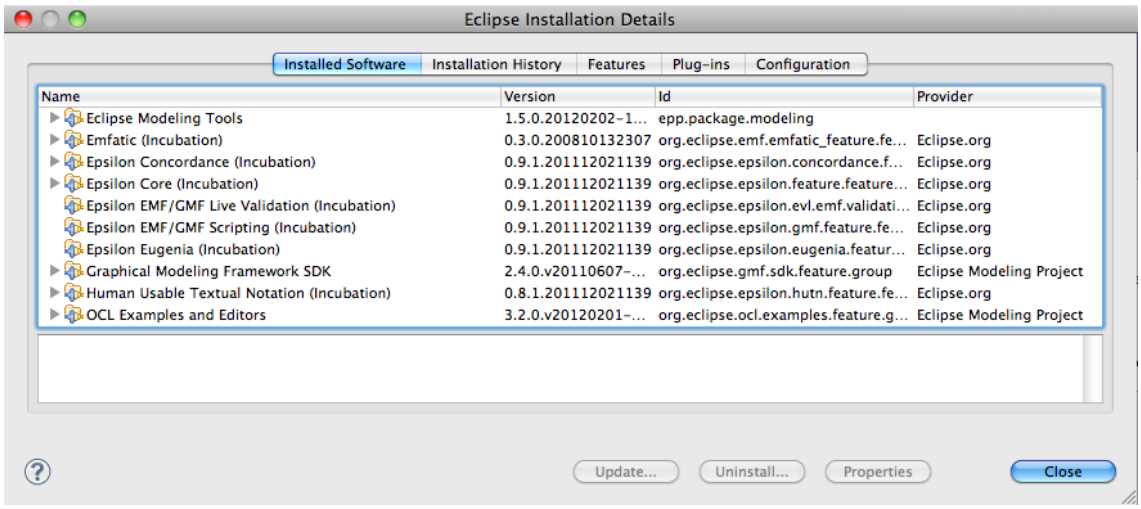


Figura C.1 Plugins necessários para a utilização do modularKAOS

Estes plugins podem ser encontrados em *Help* → *Install Modeling Components* dentro do ambiente do Eclipse, ou através do site dedicado a cada um dos plugins.

C.2 Instalar e executar o modularKAOS

Antes de instalar o modularKAOS, é necessário verificar que a versão do compilador Java é a 6.0. Para isso, vai-se a *Eclipse* → *Preferences* → *Java* → *Compiler* → *Compiler compliance level*.

Depois disto, e garantindo que o ponto anterior, da instalação dos plugins, foi feito correctamente, faz-se a instalação do modularKAOS. Esta pode ser feita de duas formas:

i. **Instalar plugins:** extrair os plugins do modularKAOS para a pasta plugin do seu Eclipse. Reinice o Eclipse para os plugins sejam carregados.

ii. **Instalar projecto:** faz-se *File* → *Import* → *General* → *Existing Projects into Workspace* → *Select archive file* → *Browse*, e indica-se o caminho para o zip do projecto. Selecciona-se as pastas correspondentes e faz-se *Finish*.

C.3 Criação de um projecto modularKAOS

Para criar um projecto modularKAOS é necessário fazer *File* → *New* → *Project*. Depois, na nova janela faz-se *General* → *Project*. Insere-se o nome do projecto e carrega-se no botão *Finish*.

Uma vez o projecto criado, deve-se criar o editor modularKAOS. Para isso, seleciona-se o projecto anteriormente definido e faz-se *File* → *New* → *Example* e seleciona-se *KAOSStandard Diagram*. Adiciona-se o nome do modelo, e carrega-se em *Finish*. Surgem dois ficheiros, um com a extensão *.kaosstandard* e outro com a extensão *.kaosstandard_diagram*. A criação do modelo vai ser feita no segundo ficheiro, e no primeiro toda a informação que vamos inserido no modelo será apresentada textualmente.

O editor gráfico é composto por uma tela, onde o modelo será desenhado, e por uma lista de elementos do lado esquerdo. Esta lista contém todos os nós e ligações que a metodologia KAOS compreende. Para validar o modelo, ou retornar as métricas associadas, seleciona-se o ficheiro *kaosstandard_diagram* e faz-se *Edit* → *Validate* ou *Metrics*.

C.4 Como efectuar modificações no modularKAOS?

Já explicamos como o utilizador comum poderá tirar partido da ferramenta modularKAOS. Resta introduzir o nosso projecto para aqueles que

pretendem extender esta ferramenta. A Figura C.2 mostra os ficheiros que podem ser encontrados no nosso projecto.

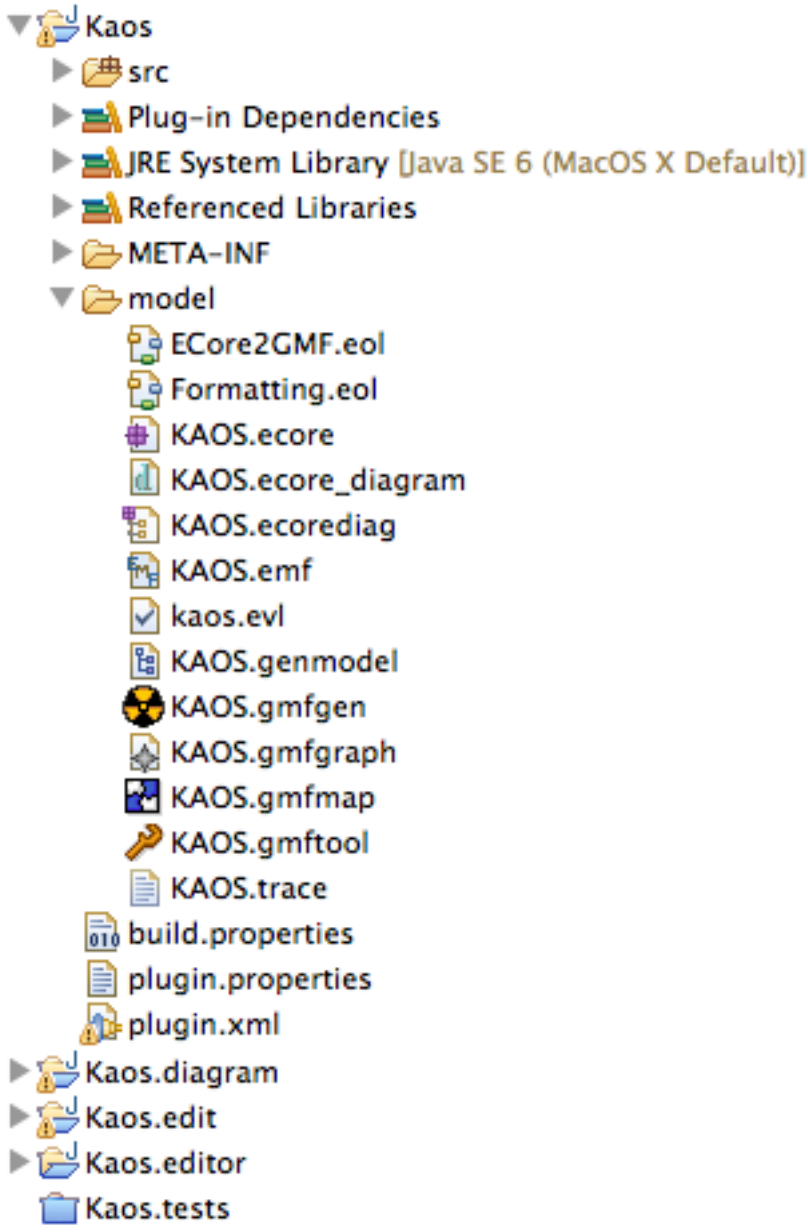


Figura C.2: Elementos do projecto modularKAOS

As alterações no projecto devem ser feitas sobre os ficheiros: *ECore2GMF.eol*, *Formatting.eol*, *KAOS.ecore*, e/ou *KAOS.emf*. Sempre que feita uma alteração nestes ficheiros, o editor gráfico tem que ser novamente gerado. Para isso, temos que garantir que os projectos *Kaos.diagram*, *Kaos.edit*, e *Kaos.editor*, estão presentes e encontram-se abertos (estes ficheiros ccontém informação essencial para o bom funcionamento do modularKAOS). Se estivermos nestas condições, podemos gerar o editor gráfico, seleccionando o

ficheiro *KAOS.emf*, com o botão direito do rato, e fazendo *Eugenia* → *Generate GMF editor*. Esta operação demora algum tempo e vai gerar: todos os ficheiros da pasta *src*; os ficheiros *KAOS.genmodel*, *KAOS.gmfgen*, *KAOS.gmfgraph*, *KAOS.gmfmap*, e *KAOS.gmftool*; e actualizar os projectos *Kaos.diagram*, *Kaos.edit*, e *Kaos.editor*. À que referir, que se as modificações forem feitas no ficheiro *ecore*, deve-se actualizar o ficheiro *emf* com estas alterações. Para isso, seleciona-se o ficheiro *KAOS.ecore,* com o botão direito do rato, e faz-se *Generate Emfatic Source*.

Depois da criação do editor gráfico, são apresentados alguns erros devido a um *bug* nos plugins EMF/GMF, até à data não resolvido pela equipa do Eclipse. Para contronar esse problema, abrimos o ficheiro *KAOS.genmodel* com um editor de texto e adicionamos a seguinte linha de código (logo a seguir à tag *genmodel:GenModel*):

```
<genAnnotations source="http://www.eclipse.org/OCL/GenModel"><details
key="Use Delegates" value="true"/></genAnnotations>
```

Posto isto, voltamos a gerar os ficheiros mas agora apartir do *KAOS.genmodel*. Abre-se este ficheiro com o editor *Emf Generator*, e clica-se sobre a raiz do ficheiro e faz-se *Generate All*. Após a criação destes novos ficheiros, basta criar uma nova instância do Eclipse para testar o editor gráfico.

O ficheiro *evl* permite inserir restrições que serão validadas no editor gráfico. As alterações neste ficheiro podem ser feitas em qualquer momento, sendo que o último editor gráfico gerado terá sempre em conta o que é definido neste ficheiro.

D

Métricas auxiliares

A Tabela D.1 apresenta algumas métricas intermédias que ajudam a compor as métricas que respondem às questões apresentadas na Tabela 4.1 e na Tabela 4.2. Cada métrica nesta tabela é composta por uma definição informal e pela sua definição formal utilizando OCL sobre o metamodelo apresentado na Figura 4.6. Sempre que o funcionamento destas métricas não for intuitivo, será adicionado um comentário para melhor explicar a respectiva métrica.

Tabela D.1: Métricas auxiliares

Nome	**EA – *E**xpectation **A**gents* (Agentes da Expectativa)
Definição informal	Conjunto de todos os agentes directamente associados a esta expectativa.
Definição formal	***context Expectation*** ***def: EA(): Set(Agent)*** = *self.expToAgentExpLink->collect(* *expLinkToAgent)->asSet()*
Nome	**RA – *R**equirement **A**gents* (Agentes do Requisito)
Definição informal	Conjunto de todos os agentes directamente associados a este requisito.
Definição formal	***context Requirement*** ***def: RA(): Set(Agent)*** = *self.ReqToAgentReqLink->collect(* *ReqToAgentLink)->asSet()*
Nome	**ILG – *I**s **L**eaf **G**oal* (É um Objectivo Folha)
Definição informal	Indica se este objectivo é um objectivo folha do modelo ou não.
Definição formal	***context Goal*** ***def: ILG(): Boolean*** = *self.goalIsAndRef->isEmpty() and*

	self.goalIsOrRef->isEmpty()
Comentários	Um objectivo é considerado uma folha do modelo caso ele não tenha refinamentos-E nem refinamentos-OU.
Nome	**GDA – *G**oal **D**irect **A**gents*** (Agentes Directos do Objectivo)
Definição informal	Conjunto de todos os agentes directamente associados a este objectivo.

Definição formal	***context Goal***
	def: GDA(): Set(Agent) **=** *if self.oclIsKindOf(Expectation) then*
	self.oclAsType(Expectation).EA()
	else
	if self.oclIsKindOf(Requirement) then
	self.oclAsType(Requirement).RA()
	else
	Set{}
	endif
	endif

Nome	**GA – *G**oal **A**gents*** (Agentes do Objectivo)
Definição informal	Conjunto de todos os agentes herdados ou directamente associados a este objectivo.

Definição formal	***context Goal***
	def: GA(visited: Set(Goal)): Set(Agent) =
	let dagents: Set(Agent) = self.GDA() in
	if visited->one(g: Goal | g = self) then
	dagents
	else
	self.otherGoalIsOrRef->collect(orRefToGoal)->union(
	self.otherGoalIsAndRef->collect(andRefToGoal)
	)->iterate(g : Goal; aux: Set(Agent) = dagents |
	aux->union(g.GA(visited->including(self)))
	)->asSet()
	endif

Comentários	As responsabilidades nos modelos de objectivos KAOS são herdadas de objectivos pais para objectivos filhos. Por exemplo, se um agente for responsável pela raiz do modelo, então, esse agente é responsável por

	qualquer outro objectivo resultante do refinamento da raiz.
Nome	**GNA – *G**oal **N**umber **A**gents*** (Número de Agentes do Objectivo)
Definição informal	Número de agentes directamente ou indirectamente associados a este objectivo.
Definição formal	***context Goal*** ***def: GNA(): Integer* = *self.GA(Set{})->size()***
Nome	**GNOp – *G**oal **N**umber **Op**erations*** (Número de Operações do Objectivo)
Definição informal	Número de operações associadas a este objectivo.
Definição formal	***context Goal*** ***def: GNOp(): Integer* = *self.goalToOperatLink->collect(operatLinkToOperationNode)->size()***
Nome	**GLG – *G**oal **L**eaf **G**oals*** (Objectivos Folha do Objectivo)
Definição informal	Conjunto de todos os objectivos folha resultantes do refinamento deste objectivo, ou seja, todos os objectivos folha que estejam associados directa ou indirectamente a este objectivo que este objectivo.
Definição formal	***context Goal*** ***def: GLG(): Set(Goal)* = *let goals: Set(Goal) = self.goalIsOrRef->collect(orRefToOtherGoal)->union(self.goalIsAndRef->collect(andRefToOtherGoal))->asSet() in goals->select(g: Goal | g.ILG())->union(goals->select(g: Goal | not g.ILG())->collect(g: Goal | g.GLG()))->asSet()***
Nome	**GSG – *G**oal **S**ub-**G**oals*** (Sub-Objectivos do Objectivo)
Definição informal	Conjunto de todos os objectivos associados a este objectivo, ou seja, objectivos que podem resultar do refinamento deste objectivo.
Definição formal	***context Goal*** ***def: GSG(visited: Goal[*]): Set(Goal)* =*** *let goals: Set(Goal) = self.goalIsOrRef->collect(orRefToOtherGoal)->union(self.goalIsAndRef->collect(andRefToOtherGoal))->asSet()->union(self.goalHasObstacle->collect(obstacle)*

	->iterate(o: Obstacle; aux: Set(Goal) = Set{} |
	aux->union(o.OS(Set{}))) in
	if visited->one(vg: Goal | vg = self) then
	goals
	else
	goals->collect(g: Goal | goals->union(
	g.GSG(visited->including(self)))->asSet()
	endif

Nome	**OpNA – *Op*eration *N*umber *A*gents** (Número de Agentes da Operação)
Definição informal	Número de agentes associados a esta operação.
Definição formal	***context OperationNode*** ***def: OpNA(): Integer*** = *self.operationIsPerformed->collect(* *perLinkToAgent)->size()*
Nome	**EAG – *E*nvironment *A*gent *G*oals** (Agentes do Ambiente do Objectivo)
Definição informal	Conjunto de todos os objectivos directamente associados a este agente do ambiente.
Definição formal	***context EnvironmentAgent*** ***def: EAG(): Set(Goal)*** = *self.agentToExpLink->collect(* *agentExpLinkToExp)->asSet()*
Nome	**SAG – *S*ystem *A*gent *G*oals** (Agentes do Sistema do Objectivo)
Definição informal	Conjunto de todos os objectivos directamente associados a este agente do sistema.
Definição formal	***context SystemAgent*** ***def: SAG(): Set(Goal)*** = *self.agentToReqLink->collect(* *agentReqLinkToReq)->asSet()*
Nome	**AG – *A*gent *G*oals** (Agentes do Objectivo)
Definição informal	Conjunto de todos os objectivos directamente associados a este agente.

Definição formal	***context Agent***
	def: AG(): Set(Goal) = *if self.oclIsKindOf(EnvironmentAgent) then*
	self.oclAsType(EnvironmentAgent).EAG()
	else
	if self.oclIsKindOf(SystemAgent) then

	self.oclAsType(SystemAgent).SAG()
	else
	Set{}
	endif
	endif

Nome	**AIG – *A**gent **I**nherited **G**oals*** (Agentes Herdados do Objectivo)
Definição informal	Conjunto de todos os objectivos indirectamente associados a este agente.
Definição formal	***context Agent*** ***def: AIG(): Set(Goal)*** *= self.AG()->select(g: Goal |* *not g.ILG())->collect(g: Goal | g.GLG())->asSet()*
Comentários	Da mesma forma que um objectivo herda agentes dos seus objectivos pais, a mesma informação deve constar nesses agentes. Portanto se um agente está associado a um objectivo, ele encontra-se associado a todos os objectivos gerados por consequência do refinamento do objectivo inicial.
Nome	**ALG – *A**gent **L**eaf **G**oals*** (Objectivos Folha do Agente)
Definição informal	Conjunto de objectivos folha directamente ou indirectamente associados a este agente.
Definição formal	***context Agent*** ***def: ALG(): Set(Goal)*** *= self.AG()->union(* *self.AIG())->select(g: Goal | g.ILG())*
Nome	**ILO – *I**s **L**eaf **O**bstacle*** (É um Obstáculo Folha)
Definição informal	Indica se este obstáculo é um obstáculo folha do modelo ou não.
Definição formal	***context Obstacle*** ***def: ILO(): Boolean*** *= self.obstIsParent->isEmpty()*
Comentários	Um obstáculo folha é um obstáculo sem refinamentos, ou seja, sem filhos.
Nome	**OR – *O**bstacle **R**efinements*** (Refinamentos do Obstáculo)
Definição informal	Conjunto de todos os obstáculos directamente associados a este obstáculo, através do seu refinamento.
Definição formal	***context Obstacle*** ***def: OR(): Set(Obstacle)*** *= self.obstIsParent->collect(* *obstRefToObst)->asSet()*
Comentários	Tal como os objectivos, os obstáculos também podem ser refinados em obstáculos mais concretos.

Nome	**OP – *O**bstacle **P**arents*** (Pais do Obstáculo)
Definição informal	Conjunto de todos os pais deste obstáculo, ou seja, todos os obstáculos em que o seu refinamento dá origem (directa) a este obstáculo.
Definição formal	***context Obstacle*** ***def: OP(): Set(Obstacle)*** *= self.obstIsRefinement->collect(* *obstToObstRef)->asSet()*
Comentários	Se os obstáculos conhecem os seus filhos, é igualmente importante que conheçam os seus pais.
Nome	**OAS – *O**bstacle **A**bove re**S**olutions*** (Resoluções de Cima do Obstáculo)
Definição informal	Conjunto de todas as resoluções que este obstáculo tem por herança dos obstáculos de um nível superior (nível mais abstracto).

Definição formal	***context Obstacle***
	def: OAS(visited: Set(Obstacle)): Set(Goal) *=*
	let sol: Set(Goal) = self.solution->collect(
	obstacleHasSolution)->asSet() in
	if visited->one(vo: Obstacle | vo = self) then
	sol
	else
	self.OP()->iterate(o: Obstacle; aux: Set(Goal) = sol |
	aux->union(o.OAS(visited->including(self))))
	endif

Nome	**OBS – *O**bstacle **B**elow re**S**olutions*** (Resoluções de Baixo do Obstáculo)
Definição informal	Conjunto de todas as resoluções que este obstáculo tem por herança dos obstáculos de um nível inferior (nível menos abstracto).

Definição formal	***context Obstacle***
	def: OBS(visited: Set(Obstacle)): Set(Goal) *=*
	let sol : Set(Goal) = self.solution->collect(obstacleHasSolution)->asSet() in
	if visited->one(vo : Obstacle | vo = self) then
	sol
	else
	if self.OR()->forAll(o : Obstacle |
	o.OS(visited->including(self))->size() > 0) then
	self.OR()->iterate(o : Obstacle; aux : Set(Goal) = sol |

	aux->union(o.OBS(visited->including(self)))
	else
	sol
	endif
	endif

Comentários	Os obstáculos apenas herdam as resoluções dos seus obstáculos filhos se todos eles tiverem pelo menos uma resolução. Só neste cenário é que é possível afirmar que as resoluções dos obstáculos filhos resolvem o obstáculo pai.
Nome	**OS – *O*bstacle re*S*olutions** (Resoluções do Obstáculo)
Definição informal	Conjunto de todas as resoluções que este obstáculo tem quer por herança, dos seus pais ou filhos, quer por associação directa.
Definição formal	***context Obstacle*** ***def: OS(visited: Set(Obstacle)): Set(Goal)*** **=** *self.OBS(visited)->union(self.OAS(visited))*
Comentários	Se um obstáculo de um nível superior de abstracção tem uma resolução, os seus filhos herdam essas mesma resoluções. E se um obstáculo resultante do refinamento deste obstáculo tem uma resolução, essa resolução é herdada por este obstáculo.
Nome	**ONS – *O*bstacle *N*umber re*S*olutions** (Número de Resoluções do Obstáculo)
Definição informal	Número de resoluções que este obstáculo tem.
Definição formal	***context Obstacle*** ***def: ONS(): Set(Goal)*** *= self.OS(Set{})->size()*

E

Aplicação do modularKAOS aos casos de estudo

Durante este capítulo, serão apresentados os resultados da aplicação da ferramenta modularKAOS a cada caso de estudo introduzido na Secção 5.1. Cada resultado, é composto por duas figuras, uma que representa o modelo de objectivos e outra os respectivos erros e avisos detectados pela ferramenta.

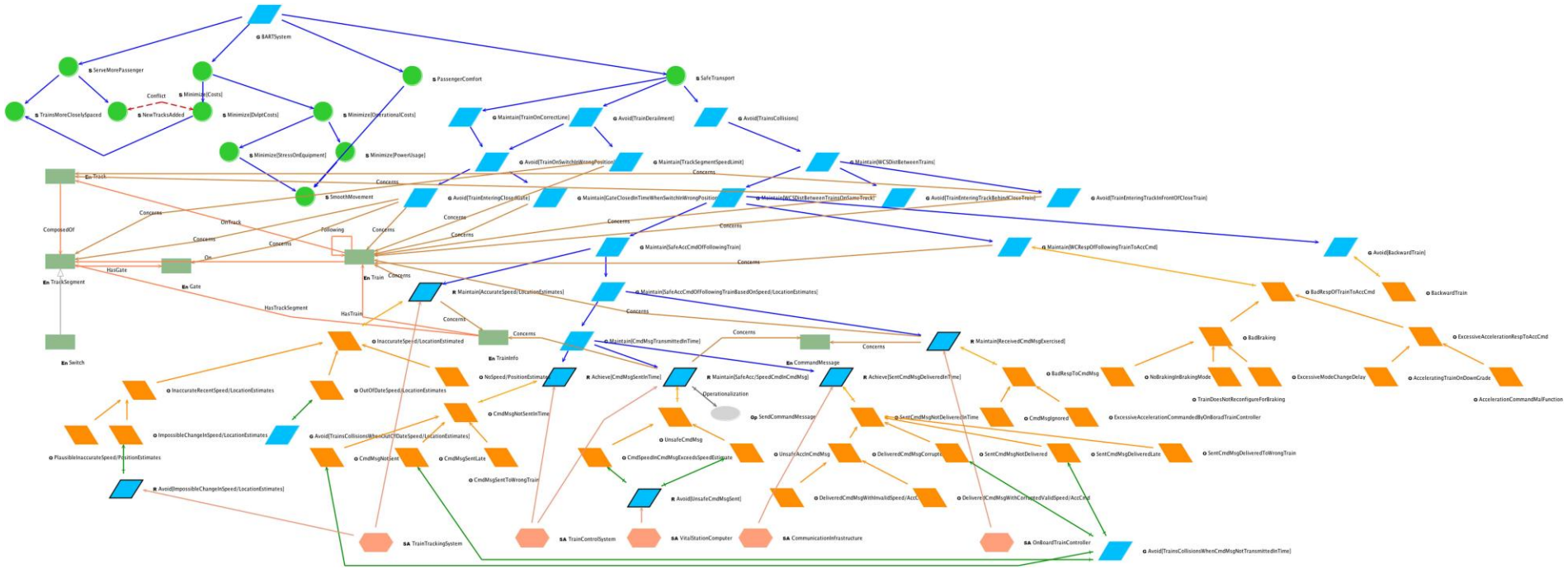


Figura E.1: Sistema BARTS modelado pela ferramenta modularKAOS

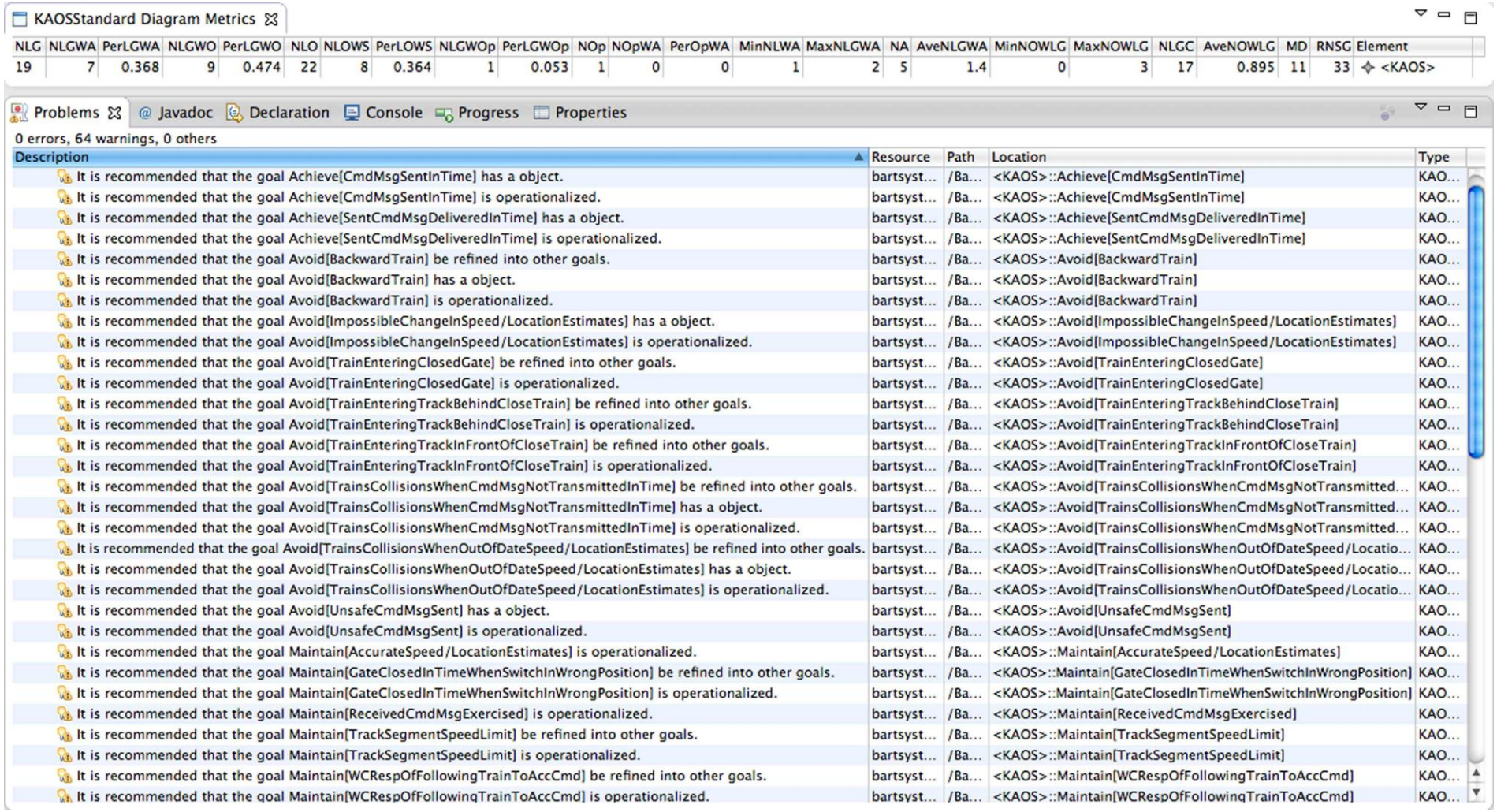


Figura E.2: Métricas e avisos do modelo do sistema BARTS

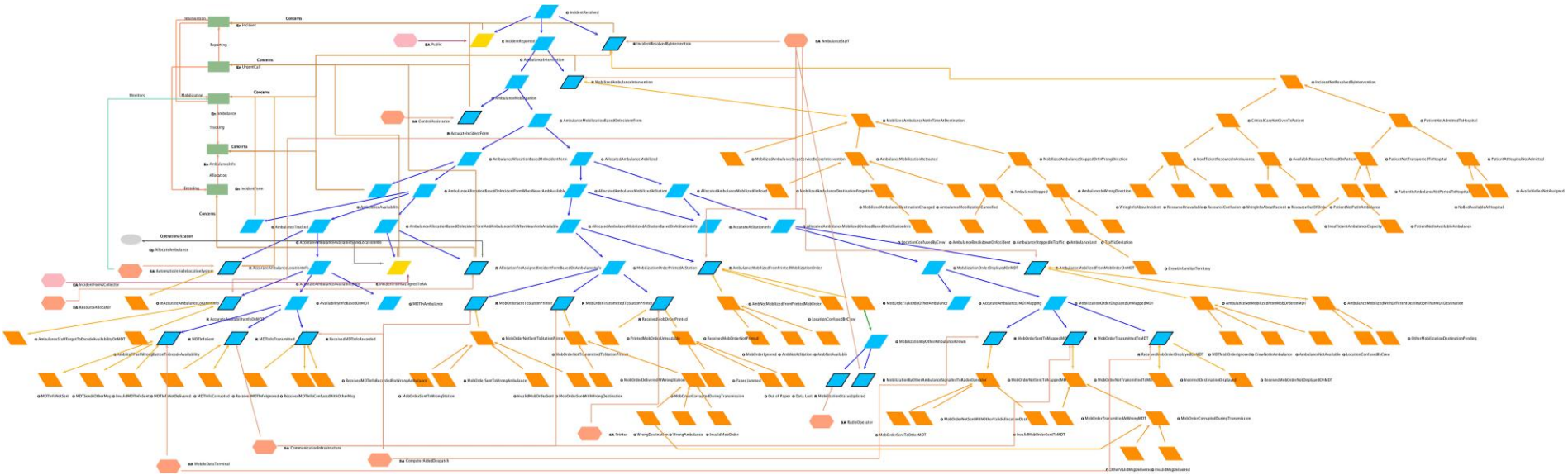
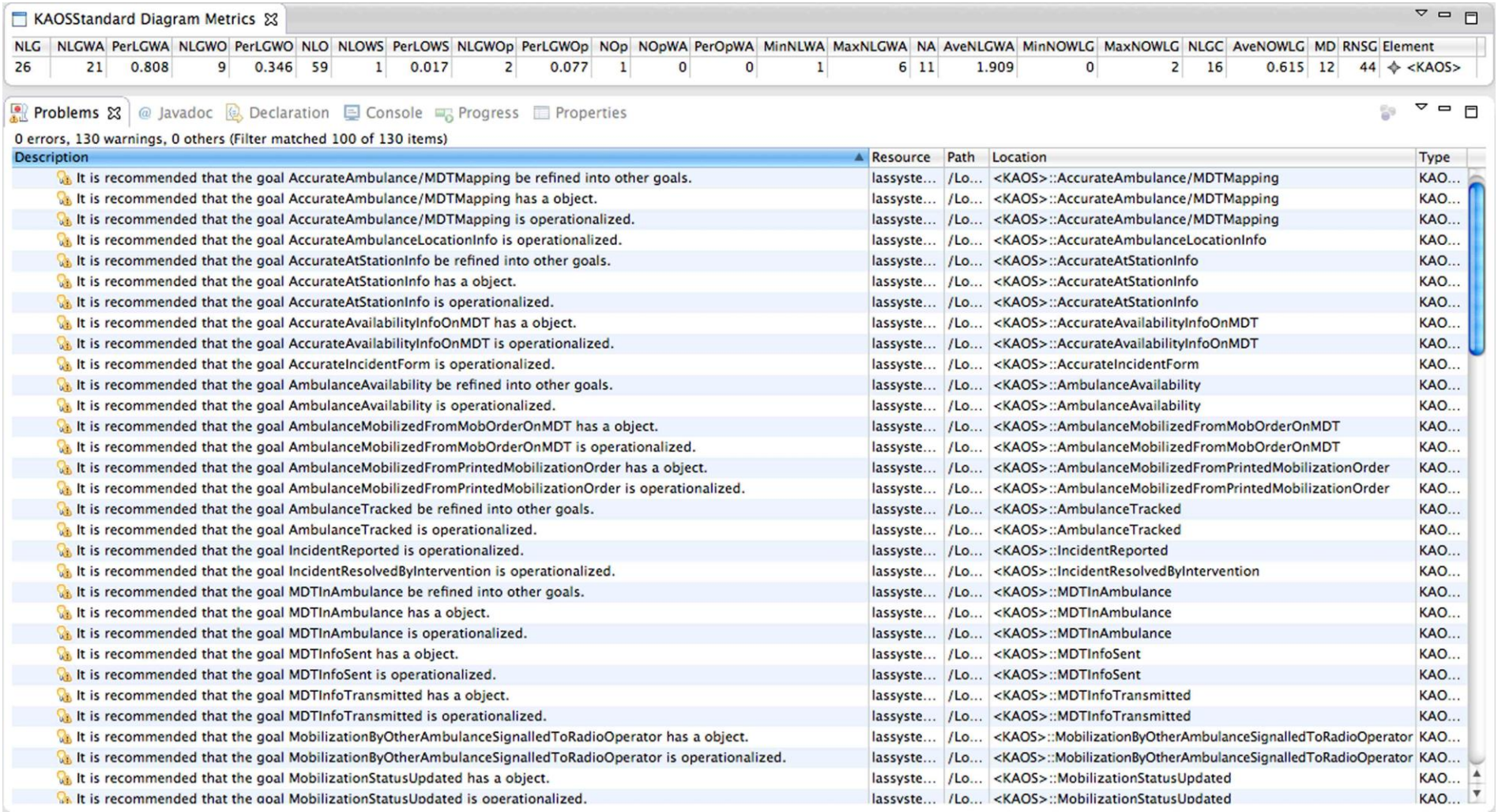


Figura E.3: Sistema LASS modelado pela ferramenta modularKAOS



KAOSStandard Diagram Metrics

NLG	NLGWA	PerLGWA	NLGWO	PerLGWO	NLO	NLOWS	PerLOWS	NLGWOp	PerLGWOp	NOp	NOpWA	PerOpWA	MinNLWA	MaxNLWA	NA	AveNLGWA	MinNOWLG	MaxNOWLG	NLGC	AveNOWLG	MD	RNSG	Element
26	21	0.808	9	0.346	59	1	0.017	2	0.077	1	0	0	1	6	11	1.909	0	2	16	0.615	12	44	<KAOS>

Problems | Javadoc | Declaration | Console | Progress | Properties

0 errors, 130 warnings, 0 others (Filter matched 100 of 130 items)

Description	Resource	Path	Location	Type
It is recommended that the goal AccurateAmbulance/MDTMapping be refined into other goals.	lassyste...	/Lo...	<KAOS>::AccurateAmbulance/MDTMapping	KAO...
It is recommended that the goal AccurateAmbulance/MDTMapping has a object.	lassyste...	/Lo...	<KAOS>::AccurateAmbulance/MDTMapping	KAO...
It is recommended that the goal AccurateAmbulance/MDTMapping is operationalized.	lassyste...	/Lo...	<KAOS>::AccurateAmbulance/MDTMapping	KAO...
It is recommended that the goal AccurateAmbulanceLocationInfo is operationalized.	lassyste...	/Lo...	<KAOS>::AccurateAmbulanceLocationInfo	KAO...
It is recommended that the goal AccurateAtStationInfo be refined into other goals.	lassyste...	/Lo...	<KAOS>::AccurateAtStationInfo	KAO...
It is recommended that the goal AccurateAtStationInfo has a object.	lassyste...	/Lo...	<KAOS>::AccurateAtStationInfo	KAO...
It is recommended that the goal AccurateAtStationInfo is operationalized.	lassyste...	/Lo...	<KAOS>::AccurateAtStationInfo	KAO...
It is recommended that the goal AccurateAvailabilityInfoOnMDT has a object.	lassyste...	/Lo...	<KAOS>::AccurateAvailabilityInfoOnMDT	KAO...
It is recommended that the goal AccurateAvailabilityInfoOnMDT is operationalized.	lassyste...	/Lo...	<KAOS>::AccurateAvailabilityInfoOnMDT	KAO...
It is recommended that the goal AccurateIncidentForm is operationalized.	lassyste...	/Lo...	<KAOS>::AccurateIncidentForm	KAO...
It is recommended that the goal AmbulanceAvailability be refined into other goals.	lassyste...	/Lo...	<KAOS>::AmbulanceAvailability	KAO...
It is recommended that the goal AmbulanceAvailability is operationalized.	lassyste...	/Lo...	<KAOS>::AmbulanceAvailability	KAO...
It is recommended that the goal AmbulanceMobilizedFromMobOrderOnMDT has a object.	lassyste...	/Lo...	<KAOS>::AmbulanceMobilizedFromMobOrderOnMDT	KAO...
It is recommended that the goal AmbulanceMobilizedFromMobOrderOnMDT is operationalized.	lassyste...	/Lo...	<KAOS>::AmbulanceMobilizedFromMobOrderOnMDT	KAO...
It is recommended that the goal AmbulanceMobilizedFromPrintedMobilizationOrder has a object.	lassyste...	/Lo...	<KAOS>::AmbulanceMobilizedFromPrintedMobilizationOrder	KAO...
It is recommended that the goal AmbulanceMobilizedFromPrintedMobilizationOrder is operationalized.	lassyste...	/Lo...	<KAOS>::AmbulanceMobilizedFromPrintedMobilizationOrder	KAO...
It is recommended that the goal AmbulanceTracked be refined into other goals.	lassyste...	/Lo...	<KAOS>::AmbulanceTracked	KAO...
It is recommended that the goal AmbulanceTracked is operationalized.	lassyste...	/Lo...	<KAOS>::AmbulanceTracked	KAO...
It is recommended that the goal IncidentReported is operationalized.	lassyste...	/Lo...	<KAOS>::IncidentReported	KAO...
It is recommended that the goal IncidentResolvedByIntervention is operationalized.	lassyste...	/Lo...	<KAOS>::IncidentResolvedByIntervention	KAO...
It is recommended that the goal MDTInAmbulance be refined into other goals.	lassyste...	/Lo...	<KAOS>::MDTInAmbulance	KAO...
It is recommended that the goal MDTInAmbulance has a object.	lassyste...	/Lo...	<KAOS>::MDTInAmbulance	KAO...
It is recommended that the goal MDTInAmbulance is operationalized.	lassyste...	/Lo...	<KAOS>::MDTInAmbulance	KAO...
It is recommended that the goal MDTInfoSent has a object.	lassyste...	/Lo...	<KAOS>::MDTInfoSent	KAO...
It is recommended that the goal MDTInfoSent is operationalized.	lassyste...	/Lo...	<KAOS>::MDTInfoSent	KAO...
It is recommended that the goal MDTInfoTransmitted has a object.	lassyste...	/Lo...	<KAOS>::MDTInfoTransmitted	KAO...
It is recommended that the goal MDTInfoTransmitted is operationalized.	lassyste...	/Lo...	<KAOS>::MDTInfoTransmitted	KAO...
It is recommended that the goal MobilizationByOtherAmbulanceSignalledToRadioOperator has a object.	lassyste...	/Lo...	<KAOS>::MobilizationByOtherAmbulanceSignalledToRadioOperator	KAO...
It is recommended that the goal MobilizationByOtherAmbulanceSignalledToRadioOperator is operationalized.	lassyste...	/Lo...	<KAOS>::MobilizationByOtherAmbulanceSignalledToRadioOperator	KAO...
It is recommended that the goal MobilizationStatusUpdated has a object.	lassyste...	/Lo...	<KAOS>::MobilizationStatusUpdated	KAO...
It is recommended that the goal MobilizationStatusUpdated is operationalized.	lassyste...	/Lo...	<KAOS>::MobilizationStatusUpdated	KAO...

Figura E.4: Métricas e avisos do modelo do sistema LASS

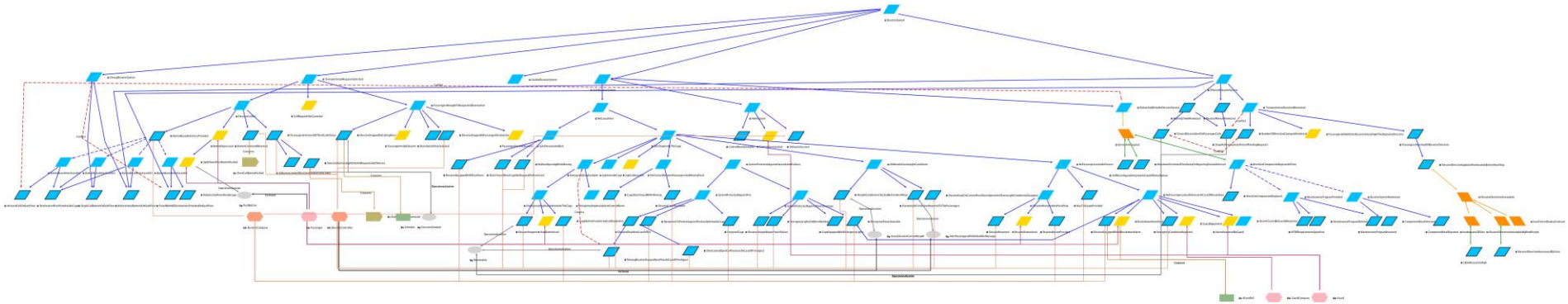


Figura E.5: Sistema ES modelado pela ferramenta modularKAOS

KAOSStandard Diagram Metrics

NLG	NLGWA	PerLGWA	NLGWO	PerLGWO	NLO	NLOWS	PerLOWS	NLGWOp	PerLGWOp	NOp	NOpWA	PerOpWA	MinNLWA	MaxNLGWA	NA	AveNLGWA	MinNOWLG	MaxNOWLG	NLGC	AveNOWLG	MD	RNSG	Element
68	41	0.603	5	0.074	4	3	0.75	9	0.132	5	3	0.6	1	18	5	8.2	0	1	5	0.074	7	101	<KAOS>

Problems | Javadoc | Declaration | Console | Progress | Properties

22 errors, 131 warnings, 0 others (Filter matched 122 of 153 items)

Description	Resource	Path	Location	Type
The goal CageEquippedWithEmergencyLights must have at least one associated agent.	esystem...	/El...	<KAOS>::CageEquippedWithEmergencyLights	KAO...
The goal CallRequestNotCanceled must have at least one associated agent.	esystem...	/El...	<KAOS>::CallRequestNotCanceled	KAO...
The goal ClosestElevatorSentOnPassengerCalls must have at least one associated agent.	esystem...	/El...	<KAOS>::ClosestElevatorSentOnPassengerCalls	KAO...
The goal ComponentWearDetected must have at least one associated agent.	esystem...	/El...	<KAOS>::ComponentWearDetected	KAO...
The goal ComponentWearReported must have at least one associated agent.	esystem...	/El...	<KAOS>::ComponentWearReported	KAO...
The goal ControlRoomLockable must have at least one associated agent.	esystem...	/El...	<KAOS>::ControlRoomLockable	KAO...
The goal DestinationFloorSelected must have at least one associated agent.	esystem...	/El...	<KAOS>::DestinationFloorSelected	KAO...
The goal ElevatorDirectionAnnouncedByVoice must have at least one associated agent.	esystem...	/El...	<KAOS>::ElevatorDirectionAnnouncedByVoice	KAO...
The goal ElevatorDirectionUpdatedFewSecondsBeforeNextStop must have at least one associated agent.	esystem...	/El...	<KAOS>::ElevatorDirectionUpdatedFewSecondsBeforeNextStop	KAO...
The goal LEDsAtLeast2inHigh must have at least one associated agent.	esystem...	/El...	<KAOS>::LEDsAtLeast2inHigh	KAO...
The goal LightsAlwaysOn must have at least one associated agent.	esystem...	/El...	<KAOS>::LightsAlwaysOn	KAO...
The goal MaintenanceProgramDefined must have at least one associated agent.	esystem...	/El...	<KAOS>::MaintenanceProgramDefined	KAO...
The goal MaintenanceProgramExecuted must have at least one associated agent.	esystem...	/El...	<KAOS>::MaintenanceProgramExecuted	KAO...
The goal MaximumPermittedTimeDelayForRepairingFailureEnforced must have at least one associated agent.	esystem...	/El...	<KAOS>::MaximumPermittedTimeDelayForRepairingFailureEnf...	KAO...
The goal MTBFRequirementsSpecified must have at least one associated agent.	esystem...	/El...	<KAOS>::MTBFRequirementsSpecified	KAO...
The goal NumberOfDirectionChangesMinimized must have at least one associated agent.	esystem...	/El...	<KAOS>::NumberOfDirectionChangesMinimized	KAO...
The goal PassengerInsideElevator must have at least one associated agent.	esystem...	/El...	<KAOS>::PassengerInsideElevator	KAO...
The goal PassengerOutsideElevator must have at least one associated agent.	esystem...	/El...	<KAOS>::PassengerOutsideElevator	KAO...
The goal PassengersDoNotEnterElevatorsHeadingInTheOppositeDirection must have at least one associated agent.	esystem...	/El...	<KAOS>::PassengersDoNotEnterElevatorsHeadingInTheOpposi...	KAO...
The goal SelfReconfigurableSystemInCaseOfPartialFailure must have at least one associated agent.	esystem...	/El...	<KAOS>::SelfReconfigurableSystemInCaseOfPartialFailure	KAO...
The goal StopsAtIntermediateFloorsIfPendingRequests must have at least one associated agent.	esystem...	/El...	<KAOS>::StopsAtIntermediateFloorsIfPendingRequests	KAO...
The goal WornOutComponentsReplaced must have at least one associated agent.	esystem...	/El...	<KAOS>::WornOutComponentsReplaced	KAO...
Warnings (100 of 131 items)				
It is recommended that the goal AlarmAnsweredByGuard has a object.	esystem...	/El...	<KAOS>::AlarmAnsweredByGuard	KAO...
It is recommended that the goal AlarmAnsweredByGuard is operationalized.	esystem...	/El...	<KAOS>::AlarmAnsweredByGuard	KAO...
It is recommended that the goal AlarmClearedByGuardsResponse be refined into other goals.	esystem...	/El...	<KAOS>::AlarmClearedByGuardsResponse	KAO...
It is recommended that the goal AlarmClearedByGuardsResponse has a object.	esystem...	/El...	<KAOS>::AlarmClearedByGuardsResponse	KAO...
It is recommended that the goal AlarmClearedByGuardsResponse is operationalized.	esystem...	/El...	<KAOS>::AlarmClearedByGuardsResponse	KAO...
It is recommended that the goal AllButtonLightsOffUntilAnElevatorGetsCalled has a object.	esystem...	/El...	<KAOS>::AllButtonLightsOffUntilAnElevatorGetsCalled	KAO...
It is recommended that the goal AllButtonLightsOffUntilAnElevatorGetsCalled is operationalized.	esystem...	/El...	<KAOS>::AllButtonLightsOffUntilAnElevatorGetsCalled	KAO...
It is recommended that the goal BidirectionalButtonsOnEachFloor has a object.	esystem...	/El...	<KAOS>::BidirectionalButtonsOnEachFloor	KAO...

Figura E.6: Métricas e avisos do modelo do sistema ES

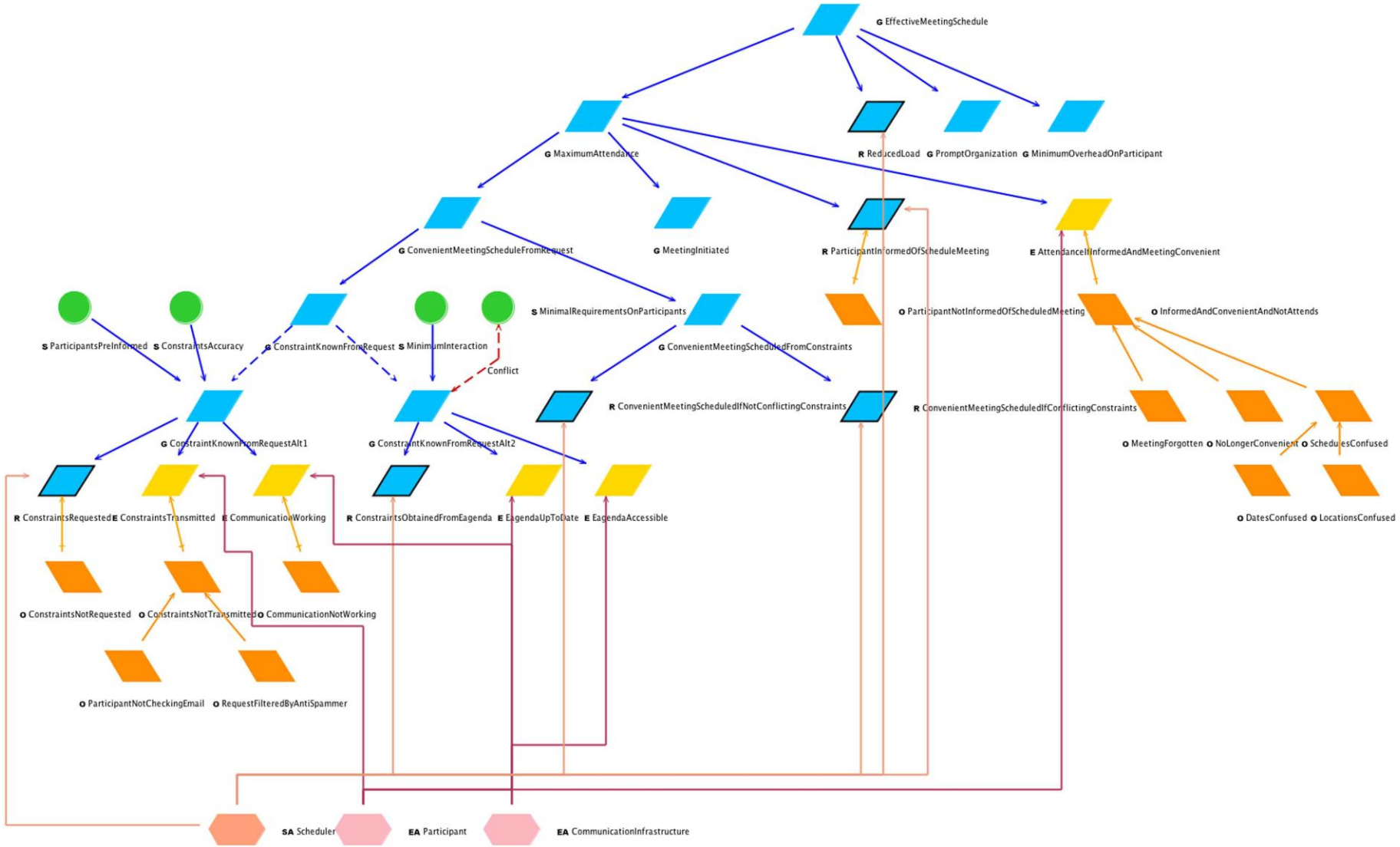
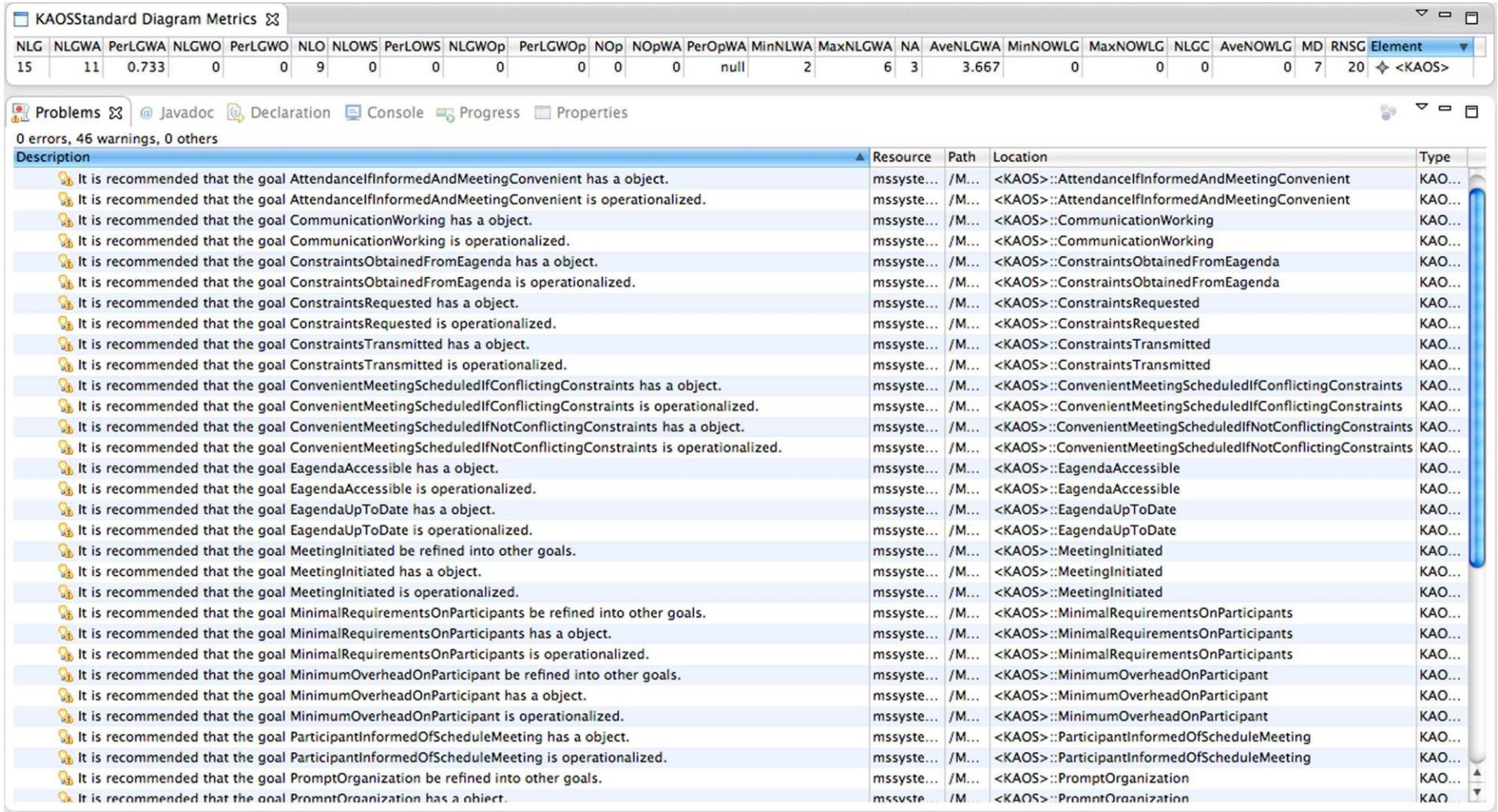


Figura E.7: Sistema MSS modelado pela ferramenta modularKAOS



KAOSStandard Diagram Metrics

NLG	NLGWA	PerLGWA	NLGWO	PerLGWO	NLO	NLOWS	PerLOWS	NLGWOp	PerLGWOp	NOp	NOpWA	PerOpWA	MinNLWA	MaxNLGWA	NA	AveNLGWA	MinNOWLG	MaxNOWLG	NLGC	AveNOWLG	MD	RNSG	Element
15	11	0.733	0	0	9	0	0	0	0	0	0	null	2	6	3	3.667	0	0	0	0	7	20	<KAOS>

Problems | Javadoc | Declaration | Console | Progress | Properties

0 errors, 46 warnings, 0 others

Description	Resource	Path	Location	Type
It is recommended that the goal AttendanceIfInformedAndMeetingConvenient has a object.	mssyste...	/M...	<KAOS>::AttendanceIfInformedAndMeetingConvenient	KAO...
It is recommended that the goal AttendanceIfInformedAndMeetingConvenient is operationalized.	mssyste...	/M...	<KAOS>::AttendanceIfInformedAndMeetingConvenient	KAO...
It is recommended that the goal CommunicationWorking has a object.	mssyste...	/M...	<KAOS>::CommunicationWorking	KAO...
It is recommended that the goal CommunicationWorking is operationalized.	mssyste...	/M...	<KAOS>::CommunicationWorking	KAO...
It is recommended that the goal ConstraintsObtainedFromEagenda has a object.	mssyste...	/M...	<KAOS>::ConstraintsObtainedFromEagenda	KAO...
It is recommended that the goal ConstraintsObtainedFromEagenda is operationalized.	mssyste...	/M...	<KAOS>::ConstraintsObtainedFromEagenda	KAO...
It is recommended that the goal ConstraintsRequested has a object.	mssyste...	/M...	<KAOS>::ConstraintsRequested	KAO...
It is recommended that the goal ConstraintsRequested is operationalized.	mssyste...	/M...	<KAOS>::ConstraintsRequested	KAO...
It is recommended that the goal ConstraintsTransmitted has a object.	mssyste...	/M...	<KAOS>::ConstraintsTransmitted	KAO...
It is recommended that the goal ConstraintsTransmitted is operationalized.	mssyste...	/M...	<KAOS>::ConstraintsTransmitted	KAO...
It is recommended that the goal ConvenientMeetingScheduledIfConflictingConstraints has a object.	mssyste...	/M...	<KAOS>::ConvenientMeetingScheduledIfConflictingConstraints	KAO...
It is recommended that the goal ConvenientMeetingScheduledIfConflictingConstraints is operationalized.	mssyste...	/M...	<KAOS>::ConvenientMeetingScheduledIfConflictingConstraints	KAO...
It is recommended that the goal ConvenientMeetingScheduledIfNotConflictingConstraints has a object.	mssyste...	/M...	<KAOS>::ConvenientMeetingScheduledIfNotConflictingConstraints	KAO...
It is recommended that the goal ConvenientMeetingScheduledIfNotConflictingConstraints is operationalized.	mssyste...	/M...	<KAOS>::ConvenientMeetingScheduledIfNotConflictingConstraints	KAO...
It is recommended that the goal EagendaAccessible has a object.	mssyste...	/M...	<KAOS>::EagendaAccessible	KAO...
It is recommended that the goal EagendaAccessible is operationalized.	mssyste...	/M...	<KAOS>::EagendaAccessible	KAO...
It is recommended that the goal EagendaUpToDate has a object.	mssyste...	/M...	<KAOS>::EagendaUpToDate	KAO...
It is recommended that the goal EagendaUpToDate is operationalized.	mssyste...	/M...	<KAOS>::EagendaUpToDate	KAO...
It is recommended that the goal MeetingInitiated be refined into other goals.	mssyste...	/M...	<KAOS>::MeetingInitiated	KAO...
It is recommended that the goal MeetingInitiated has a object.	mssyste...	/M...	<KAOS>::MeetingInitiated	KAO...
It is recommended that the goal MeetingInitiated is operationalized.	mssyste...	/M...	<KAOS>::MeetingInitiated	KAO...
It is recommended that the goal MinimalRequirementsOnParticipants be refined into other goals.	mssyste...	/M...	<KAOS>::MinimalRequirementsOnParticipants	KAO...
It is recommended that the goal MinimalRequirementsOnParticipants has a object.	mssyste...	/M...	<KAOS>::MinimalRequirementsOnParticipants	KAO...
It is recommended that the goal MinimalRequirementsOnParticipants is operationalized.	mssyste...	/M...	<KAOS>::MinimalRequirementsOnParticipants	KAO...
It is recommended that the goal MinimumOverheadOnParticipant be refined into other goals.	mssyste...	/M...	<KAOS>::MinimumOverheadOnParticipant	KAO...
It is recommended that the goal MinimumOverheadOnParticipant has a object.	mssyste...	/M...	<KAOS>::MinimumOverheadOnParticipant	KAO...
It is recommended that the goal MinimumOverheadOnParticipant is operationalized.	mssyste...	/M...	<KAOS>::MinimumOverheadOnParticipant	KAO...
It is recommended that the goal ParticipantInformedOfScheduleMeeting has a object.	mssyste...	/M...	<KAOS>::ParticipantInformedOfScheduleMeeting	KAO...
It is recommended that the goal ParticipantInformedOfScheduleMeeting is operationalized.	mssyste...	/M...	<KAOS>::ParticipantInformedOfScheduleMeeting	KAO...
It is recommended that the goal PromptOrganization be refined into other goals.	mssyste...	/M...	<KAOS>::PromptOrganization	KAO...
It is recommended that the goal PromptOrganization has a object.	mssyste...	/M...	<KAOS>::PromptOrganization	KAO...

Figura E.8: Métricas e avisos do modelo do sistema MSS

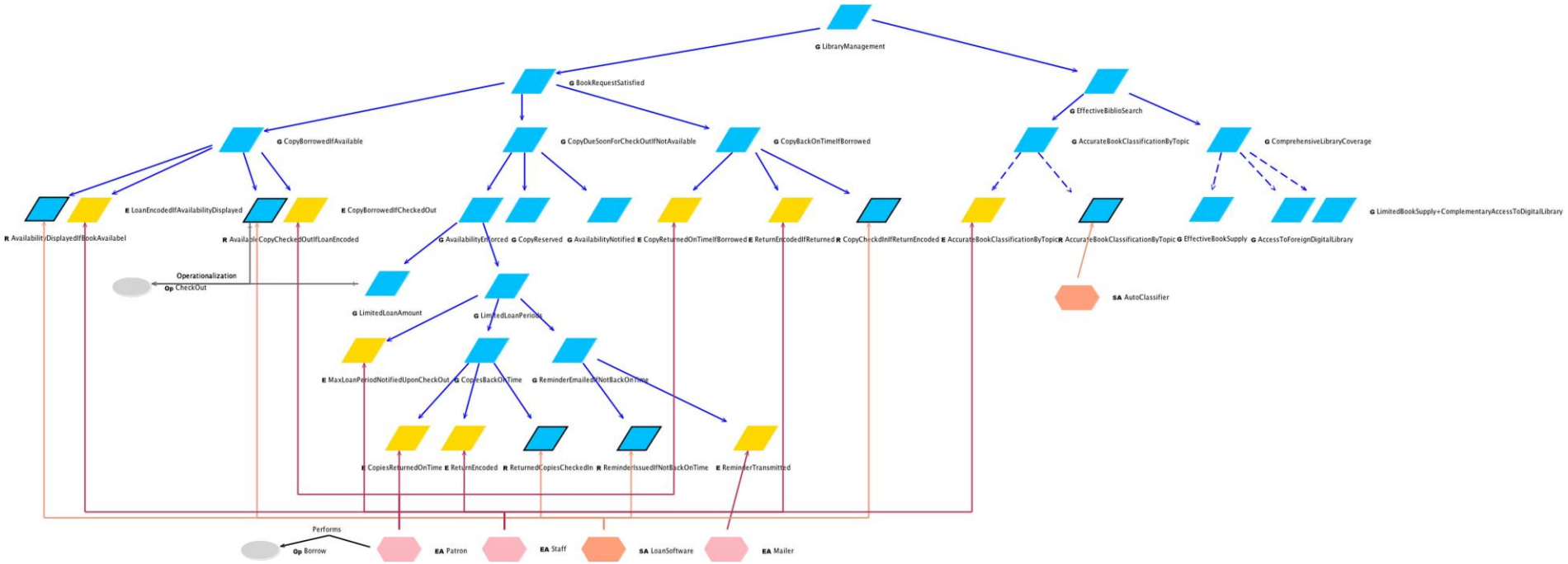
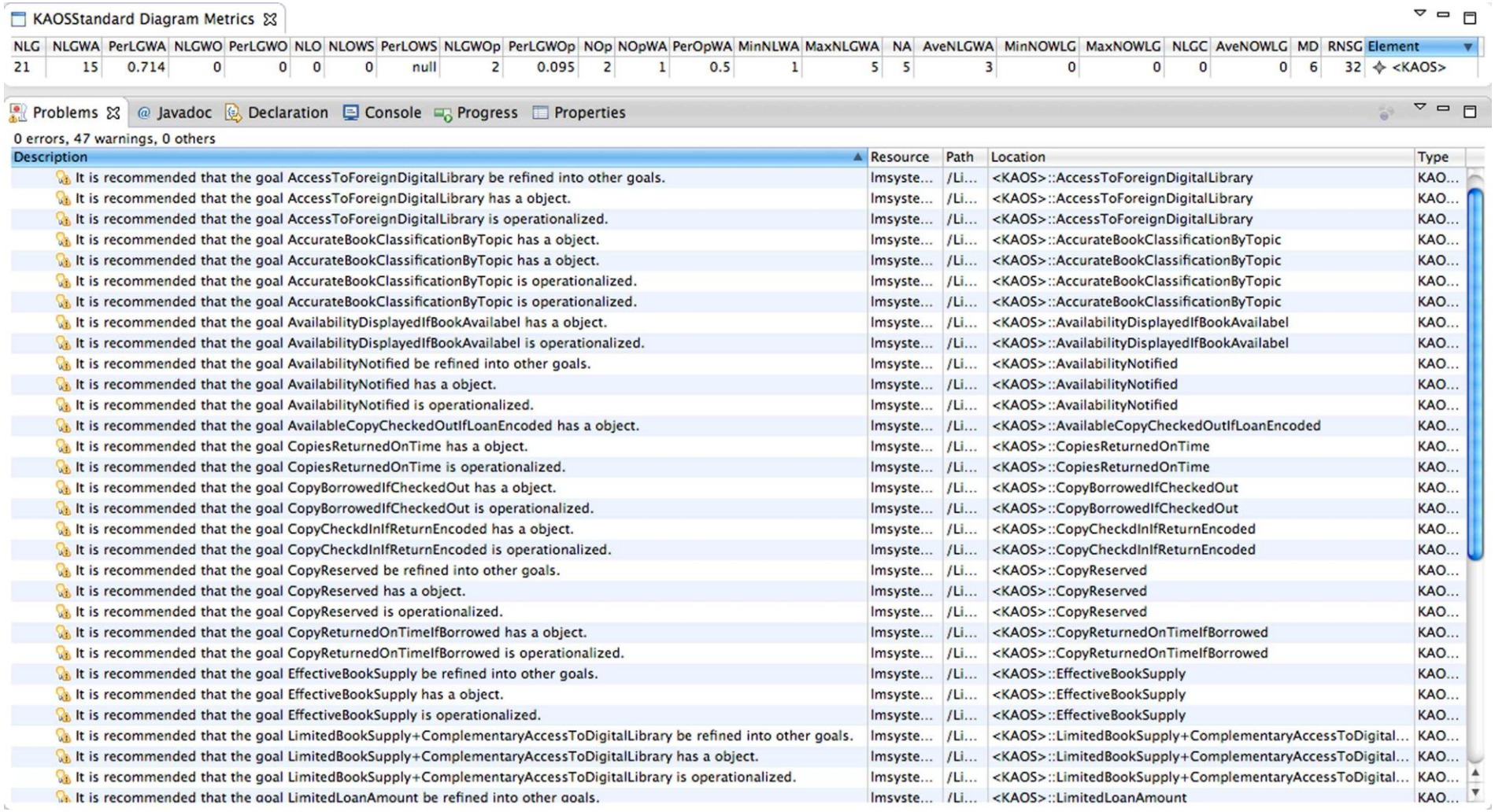


Figura E.9: Sistema LMS modelado pela ferramenta modularKAOS



KAOSStandard Diagram Metrics

NLG	NLGWA	PerLGWA	NLGWO	PerLGWO	NLO	NLOWS	PerLOWS	NLGWOp	PerLGWOp	NOp	NOpWA	PerOpWA	MinNLWA	MaxNLGWA	NA	AveNLGWA	MinNOWLG	MaxNOWLG	NLGC	AveNOWLG	MD	RNSG	Element
21	15	0.714	0	0	0	0	null	2	0.095	2	1	0.5	1	5	5	3	0	0	0	0	6	32	<KAOS>

Problems | Javadoc | Declaration | Console | Progress | Properties

0 errors, 47 warnings, 0 others

Description	Resource	Path	Location	Type
It is recommended that the goal AccessToForeignDigitalLibrary be refined into other goals.	lmsyste...	/Li...	<KAOS>::AccessToForeignDigitalLibrary	KAO...
It is recommended that the goal AccessToForeignDigitalLibrary has a object.	lmsyste...	/Li...	<KAOS>::AccessToForeignDigitalLibrary	KAO...
It is recommended that the goal AccessToForeignDigitalLibrary is operationalized.	lmsyste...	/Li...	<KAOS>::AccessToForeignDigitalLibrary	KAO...
It is recommended that the goal AccurateBookClassificationByTopic has a object.	lmsyste...	/Li...	<KAOS>::AccurateBookClassificationByTopic	KAO...
It is recommended that the goal AccurateBookClassificationByTopic has a object.	lmsyste...	/Li...	<KAOS>::AccurateBookClassificationByTopic	KAO...
It is recommended that the goal AccurateBookClassificationByTopic is operationalized.	lmsyste...	/Li...	<KAOS>::AccurateBookClassificationByTopic	KAO...
It is recommended that the goal AccurateBookClassificationByTopic is operationalized.	lmsyste...	/Li...	<KAOS>::AccurateBookClassificationByTopic	KAO...
It is recommended that the goal AvailabilityDisplayedIfBookAvailabel has a object.	lmsyste...	/Li...	<KAOS>::AvailabilityDisplayedIfBookAvailabel	KAO...
It is recommended that the goal AvailabilityDisplayedIfBookAvailabel is operationalized.	lmsyste...	/Li...	<KAOS>::AvailabilityDisplayedIfBookAvailabel	KAO...
It is recommended that the goal AvailabilityNotified be refined into other goals.	lmsyste...	/Li...	<KAOS>::AvailabilityNotified	KAO...
It is recommended that the goal AvailabilityNotified has a object.	lmsyste...	/Li...	<KAOS>::AvailabilityNotified	KAO...
It is recommended that the goal AvailabilityNotified is operationalized.	lmsyste...	/Li...	<KAOS>::AvailabilityNotified	KAO...
It is recommended that the goal AvailableCopyCheckedOutIfLoanEncoded has a object.	lmsyste...	/Li...	<KAOS>::AvailableCopyCheckedOutIfLoanEncoded	KAO...
It is recommended that the goal CopiesReturnedOnTime has a object.	lmsyste...	/Li...	<KAOS>::CopiesReturnedOnTime	KAO...
It is recommended that the goal CopiesReturnedOnTime is operationalized.	lmsyste...	/Li...	<KAOS>::CopiesReturnedOnTime	KAO...
It is recommended that the goal CopyBorrowedIfCheckedOut has a object.	lmsyste...	/Li...	<KAOS>::CopyBorrowedIfCheckedOut	KAO...
It is recommended that the goal CopyBorrowedIfCheckedOut is operationalized.	lmsyste...	/Li...	<KAOS>::CopyBorrowedIfCheckedOut	KAO...
It is recommended that the goal CopyCheckdInIfReturnEncoded has a object.	lmsyste...	/Li...	<KAOS>::CopyCheckdInIfReturnEncoded	KAO...
It is recommended that the goal CopyCheckdInIfReturnEncoded is operationalized.	lmsyste...	/Li...	<KAOS>::CopyCheckdInIfReturnEncoded	KAO...
It is recommended that the goal CopyReserved be refined into other goals.	lmsyste...	/Li...	<KAOS>::CopyReserved	KAO...
It is recommended that the goal CopyReserved has a object.	lmsyste...	/Li...	<KAOS>::CopyReserved	KAO...
It is recommended that the goal CopyReserved is operationalized.	lmsyste...	/Li...	<KAOS>::CopyReserved	KAO...
It is recommended that the goal CopyReturnedOnTimeIfBorrowed has a object.	lmsyste...	/Li...	<KAOS>::CopyReturnedOnTimeIfBorrowed	KAO...
It is recommended that the goal CopyReturnedOnTimeIfBorrowed is operationalized.	lmsyste...	/Li...	<KAOS>::CopyReturnedOnTimeIfBorrowed	KAO...
It is recommended that the goal EffectiveBookSupply be refined into other goals.	lmsyste...	/Li...	<KAOS>::EffectiveBookSupply	KAO...
It is recommended that the goal EffectiveBookSupply has a object.	lmsyste...	/Li...	<KAOS>::EffectiveBookSupply	KAO...
It is recommended that the goal EffectiveBookSupply is operationalized.	lmsyste...	/Li...	<KAOS>::EffectiveBookSupply	KAO...
It is recommended that the goal LimitedBookSupply+ComplementaryAccessToDigitalLibrary be refined into other goals.	lmsyste...	/Li...	<KAOS>::LimitedBookSupply+ComplementaryAccessToDigital...	KAO...
It is recommended that the goal LimitedBookSupply+ComplementaryAccessToDigitalLibrary has a object.	lmsyste...	/Li...	<KAOS>::LimitedBookSupply+ComplementaryAccessToDigital...	KAO...
It is recommended that the goal LimitedBookSupply+ComplementaryAccessToDigitalLibrary is operationalized.	lmsyste...	/Li...	<KAOS>::LimitedBookSupply+ComplementaryAccessToDigital...	KAO...
It is recommended that the goal LimitedLoanAmount be refined into other goals.	lmsyste...	/Li...	<KAOS>::LimitedLoanAmount	KAO...

Figura E.10: Métricas e avisos do modelo do sistema LMS

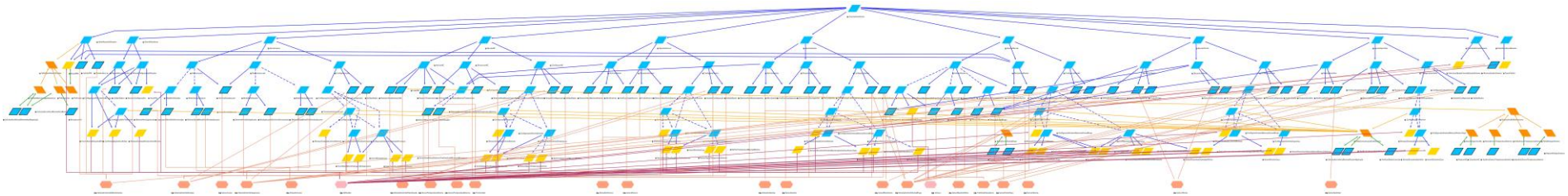


Figura E.11: Sistema SHS modelado pela ferramenta modularKAOS

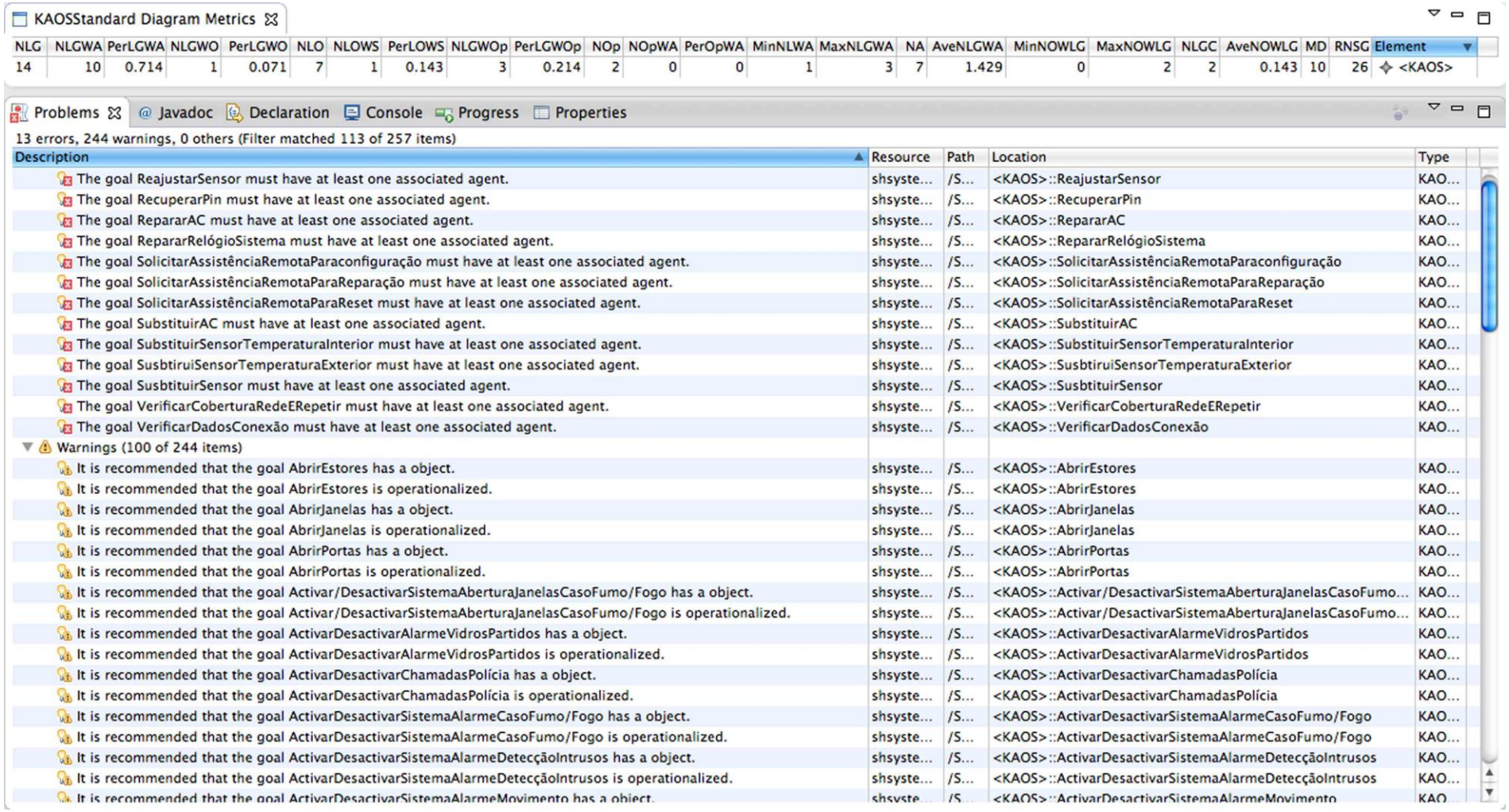


Figura E.12: Métricas e avisos do modelo do sistema SHS

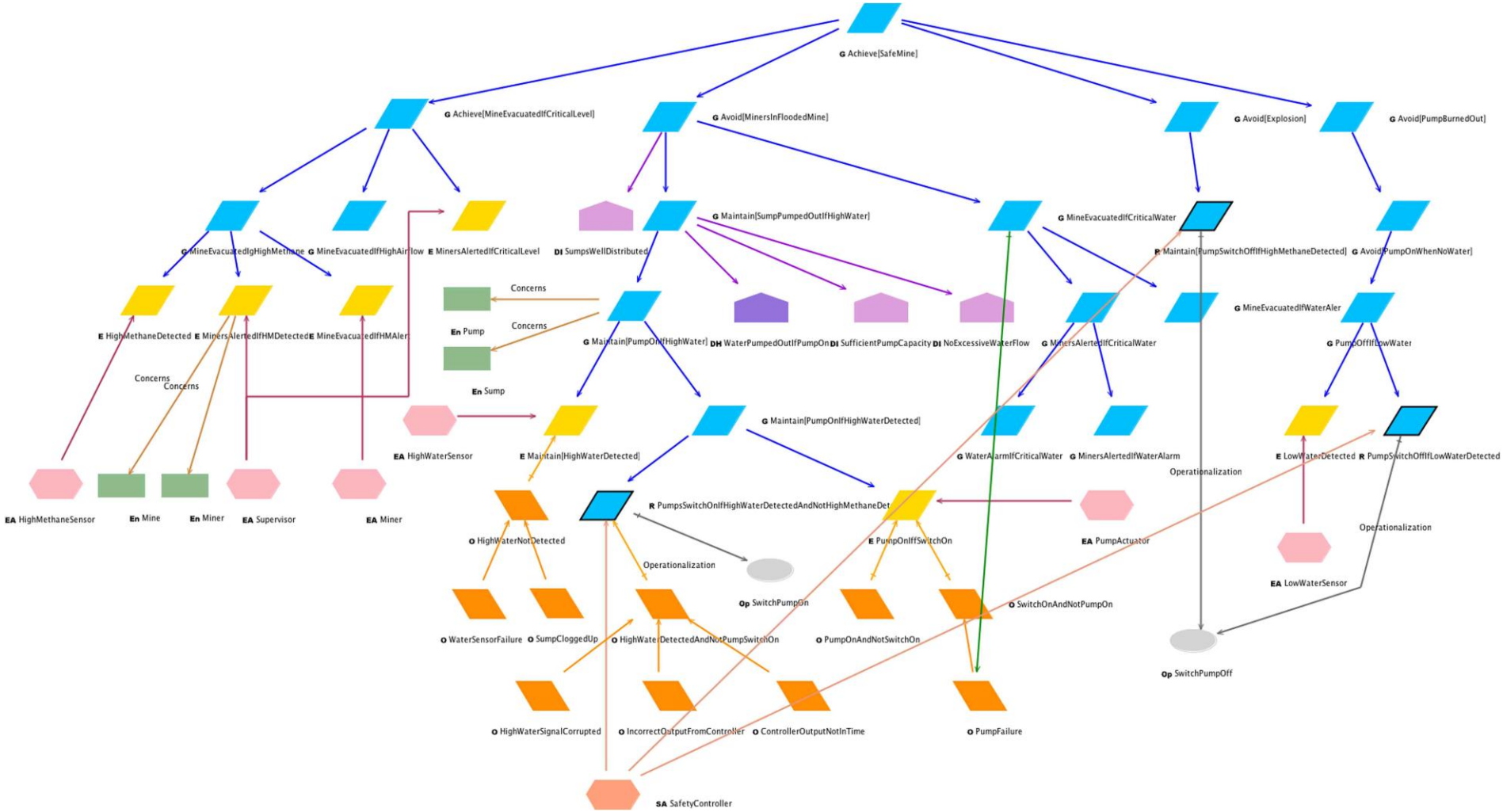
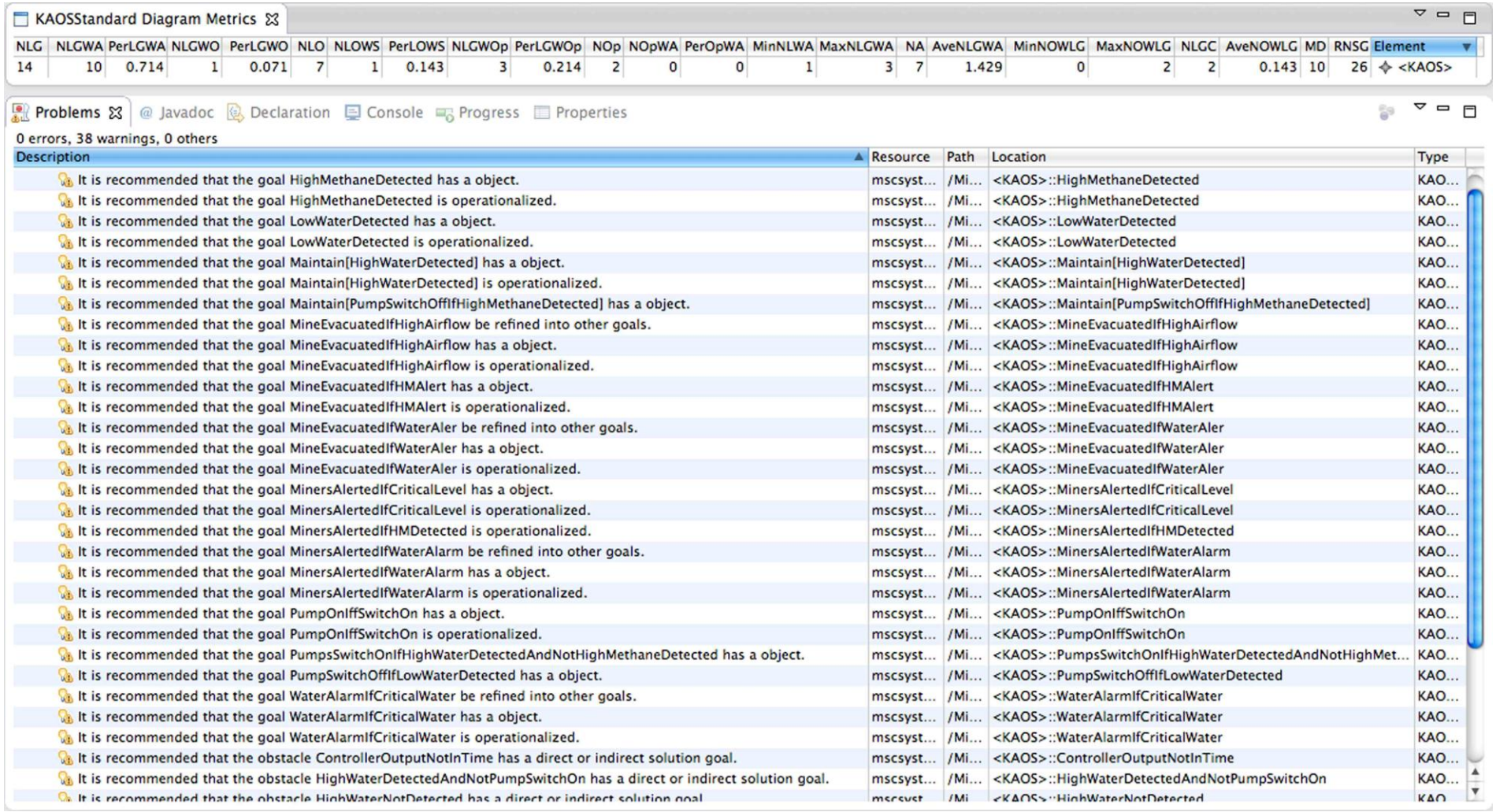


Figura E.13: Sistema MSCS modelado pela ferramenta modularKAOS



KAOSStandard Diagram Metrics

NLG	NLGWA	PerLGWA	NLGWO	PerLGWO	NLO	NLOWS	PerLOWS	NLGWOp	PerLGWOp	NOp	NOpWA	PerOpWA	MinNLWA	MaxNLGWA	NA	AveNLGWA	MinNOWLG	MaxNOWLG	NLGC	AveNOWLG	MD	RNSG	Element
14	10	0.714	1	0.071	7	1	0.143	3	0.214	2	0	0	1	3	7	1.429	0	2	2	0.143	10	26	<KAOS>

Problems | Javadoc | Declaration | Console | Progress | Properties

0 errors, 38 warnings, 0 others

Description	Resource	Path	Location	Type
It is recommended that the goal HighMethaneDetected has a object.	mscsyst...	/Mi...	<KAOS>::HighMethaneDetected	KAO...
It is recommended that the goal HighMethaneDetected is operationalized.	mscsyst...	/Mi...	<KAOS>::HighMethaneDetected	KAO...
It is recommended that the goal LowWaterDetected has a object.	mscsyst...	/Mi...	<KAOS>::LowWaterDetected	KAO...
It is recommended that the goal LowWaterDetected is operationalized.	mscsyst...	/Mi...	<KAOS>::LowWaterDetected	KAO...
It is recommended that the goal Maintain[HighWaterDetected] has a object.	mscsyst...	/Mi...	<KAOS>::Maintain[HighWaterDetected]	KAO...
It is recommended that the goal Maintain[HighWaterDetected] is operationalized.	mscsyst...	/Mi...	<KAOS>::Maintain[HighWaterDetected]	KAO...
It is recommended that the goal Maintain[PumpSwitchOffIfHighMethaneDetected] has a object.	mscsyst...	/Mi...	<KAOS>::Maintain[PumpSwitchOffIfHighMethaneDetected]	KAO...
It is recommended that the goal MineEvacuatedIfHighAirflow be refined into other goals.	mscsyst...	/Mi...	<KAOS>::MineEvacuatedIfHighAirflow	KAO...
It is recommended that the goal MineEvacuatedIfHighAirflow has a object.	mscsyst...	/Mi...	<KAOS>::MineEvacuatedIfHighAirflow	KAO...
It is recommended that the goal MineEvacuatedIfHighAirflow is operationalized.	mscsyst...	/Mi...	<KAOS>::MineEvacuatedIfHighAirflow	KAO...
It is recommended that the goal MineEvacuatedIfHMAlert has a object.	mscsyst...	/Mi...	<KAOS>::MineEvacuatedIfHMAlert	KAO...
It is recommended that the goal MineEvacuatedIfHMAlert is operationalized.	mscsyst...	/Mi...	<KAOS>::MineEvacuatedIfHMAlert	KAO...
It is recommended that the goal MineEvacuatedIfWaterAler be refined into other goals.	mscsyst...	/Mi...	<KAOS>::MineEvacuatedIfWaterAler	KAO...
It is recommended that the goal MineEvacuatedIfWaterAler has a object.	mscsyst...	/Mi...	<KAOS>::MineEvacuatedIfWaterAler	KAO...
It is recommended that the goal MineEvacuatedIfWaterAler is operationalized.	mscsyst...	/Mi...	<KAOS>::MineEvacuatedIfWaterAler	KAO...
It is recommended that the goal MinersAlertedIfCriticalLevel has a object.	mscsyst...	/Mi...	<KAOS>::MinersAlertedIfCriticalLevel	KAO...
It is recommended that the goal MinersAlertedIfCriticalLevel is operationalized.	mscsyst...	/Mi...	<KAOS>::MinersAlertedIfCriticalLevel	KAO...
It is recommended that the goal MinersAlertedIfHMDetected is operationalized.	mscsyst...	/Mi...	<KAOS>::MinersAlertedIfHMDetected	KAO...
It is recommended that the goal MinersAlertedIfWaterAlarm be refined into other goals.	mscsyst...	/Mi...	<KAOS>::MinersAlertedIfWaterAlarm	KAO...
It is recommended that the goal MinersAlertedIfWaterAlarm has a object.	mscsyst...	/Mi...	<KAOS>::MinersAlertedIfWaterAlarm	KAO...
It is recommended that the goal MinersAlertedIfWaterAlarm is operationalized.	mscsyst...	/Mi...	<KAOS>::MinersAlertedIfWaterAlarm	KAO...
It is recommended that the goal PumpOnIffSwitchOn has a object.	mscsyst...	/Mi...	<KAOS>::PumpOnIffSwitchOn	KAO...
It is recommended that the goal PumpOnIffSwitchOn is operationalized.	mscsyst...	/Mi...	<KAOS>::PumpOnIffSwitchOn	KAO...
It is recommended that the goal PumpsSwitchOnIfHighWaterDetectedAndNotHighMethaneDetected has a object.	mscsyst...	/Mi...	<KAOS>::PumpsSwitchOnIfHighWaterDetectedAndNotHighMet...	KAO...
It is recommended that the goal PumpSwitchOffIfLowWaterDetected has a object.	mscsyst...	/Mi...	<KAOS>::PumpSwitchOffIfLowWaterDetected	KAO...
It is recommended that the goal WaterAlarmIfCriticalWater be refined into other goals.	mscsyst...	/Mi...	<KAOS>::WaterAlarmIfCriticalWater	KAO...
It is recommended that the goal WaterAlarmIfCriticalWater has a object.	mscsyst...	/Mi...	<KAOS>::WaterAlarmIfCriticalWater	KAO...
It is recommended that the goal WaterAlarmIfCriticalWater is operationalized.	mscsyst...	/Mi...	<KAOS>::WaterAlarmIfCriticalWater	KAO...
It is recommended that the obstacle ControllerOutputNotInTime has a direct or indirect solution goal.	mscsyst...	/Mi...	<KAOS>::ControllerOutputNotInTime	KAO...
It is recommended that the obstacle HighWaterDetectedAndNotPumpSwitchOn has a direct or indirect solution goal.	mscsyst...	/Mi...	<KAOS>::HighWaterDetectedAndNotPumpSwitchOn	KAO...
It is recommended that the obstacle HighWaterNotDetected has a direct or indirect solution goal.	mscsyst...	/Mi...	<KAOS>::HighWaterNotDetected	KAO...

Figura E.14: Métricas e avisos do modelo do sistema MSCS